UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITECNICA SUPERIOR
INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

“Sensor Real Time”

Monitorización de redes de sensores inalámbricas a través de dispositivos Android

Autor: Isidro Muñoz Sánchez
Tutor: Francisco Javier Ordóñez Morales
Mayo 2013

Tabla de contenidos

Índice de ilustraciones

Índice de tablas

Capítulo 1

INTRODUCCIÓN

Cuando en 1992 la *join venture* entre IBM y BellSouth presentaba en sociedad el IBM Simon [1], el que es considerado como el primer *smartphone*, un teléfono que incorporaba servicios de voz y datos además de un asistente digital personal, nadie podía pensar la evolución que dichos terminales iban a tener en los años posteriores.

Aunque ya en 1992 existiese un *smartphone* con pantalla táctil, teclado virtual *QWERTY* y sin botones físicos, la popularización de estos dispositivos no llegaría hasta años más tarde, cuando el 29 de Junio de 2007 Apple empezase a comercializar su *iPhone*, un dispositivo que podía funcionar como videocámara, cámara de fotos, navegador de Internet, GPS, reproductor multimedia, además de lógicamente poder funcionar como teléfono. La llegada del iPhone ocasionó también la llegada posteriormente de la App Store, un servicio que permite a los usuarios buscar y descargar aplicaciones, esta tienda de aplicaciones permite a los desarrolladores de aplicaciones acercase de una forma sencilla a los usuarios finales. Durante los últimos años han ido en aumento tanto las aplicaciones disponibles como la descarga de las mismas. En la ilustración 1 se muestra un gráfico con las cifras:

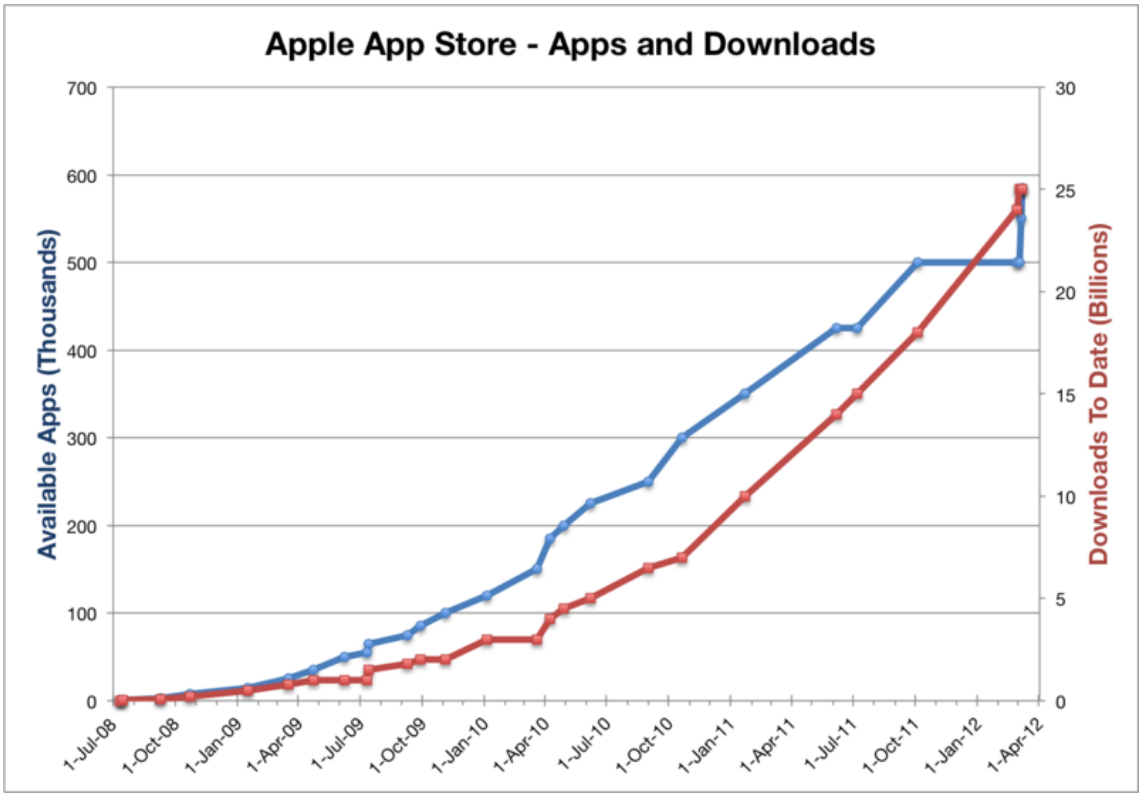


Ilustración 1: App Store

Si Apple fue el encargado de redefinir los *smartphone*, Google mediante Android ha sido el encargado de acercar los *smartphones* al público en general haciendo que distintos fabricantes puedan comercializar *smartphones* a un precio más asequible que los iPhone. El crecimiento de la plataforma de Android ha sido constante desde el lanzamiento de la plataforma. A continuación se puede ver el aumento espectacular de dispositivos con Android en Estados Unidos [3]

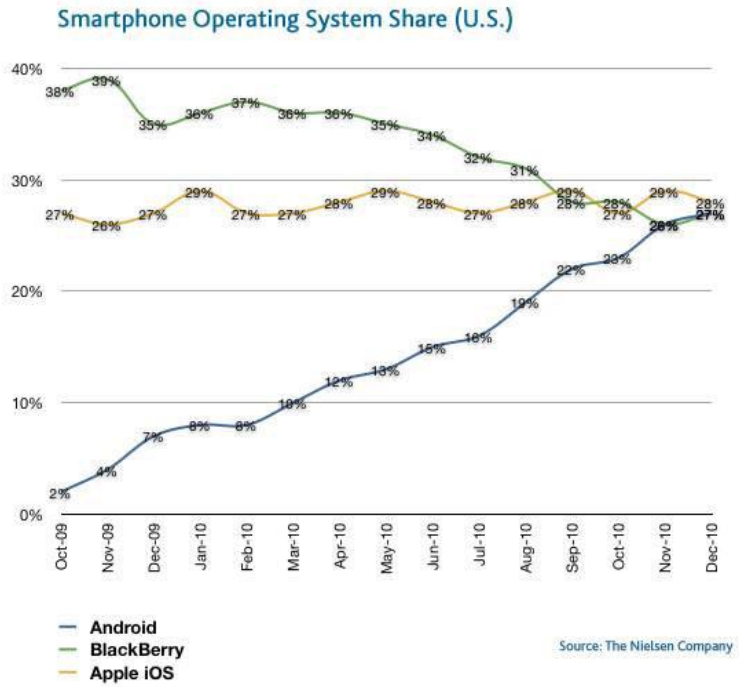


Ilustración 2: Evolución plataforma Android

De forma análoga al crecimiento del App Store, también ha ido creciendo la tienda de aplicaciones de Android, Google Play.

Al igual que ha pasado con los *smartphones* la telemedicina, que se define como el uso de las tecnologías de la información y las telecomunicaciones para el diagnóstico médico y cuidado de pacientes, ofreciendo un servicio cómodo y mejorando la calidad de vida de los usuarios, ha sufrido grandes avances con el paso de los años.

Cada día las personas demandan más aplicaciones que le permitan en tiempo real disponer de la información que precisan, ya sea información del tiempo, información del tráfico, información de resultados deportivos, de noticias… y por supuesto también de aplicaciones orientadas a la telemedicina como es el objetivo de este proyecto. El objetivo del proyecto es el diseño e implementación de un sistema de comunicación en tiempo real mediante tecnología *push* para el sistema operativo Android, aplicado a la monitorización de un entorno doméstico.

Las alternativas a la hora de realizar una aplicación con notificaciones *push* son realmente escasas debido a que no hay muchas soluciones disponibles para el entorno de Android que permitan realizar los objetivos propuestos en este proyecto, dentro de las soluciones posibles se encuentra Cloud to Device Messaging (C2DM) y MQTT, esta última solución ha sido la elegida para llevar a cabo el proyecto debido a su orientación al envío de una gran cantidad de números de mensajes.

Después de una breve introducción, a continuación se detalla la estructura del resto del documento:

1. INTRODUCCIÓN: Se realizará un pequeño resumen del universo de la aplicación, estructura de la memoria, objetivos primarios…

2. GESTIÓN Y ORGANIZACIÓN DEL PROYECTO: Se tratarán todos los temas relativos a la metodología que se va a seguir en la ejecución de este proyecto.

3. ESTADO DEL ARTE: Se analizarán las últimas novedades y recientes desarrollos en el mundo tecnológico actual que tengan algún tipo de relación con el uso de terminales móviles.

4. OBJETIVOS: Se enumerarán los objetivos de este Proyecto Fin de Carrera, tanto el objetivo general como los específicos.

5. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN: Se detallará el trabajo realizado ofreciendo una descripción de los requisitos, arquitectura y creación de la aplicación.

6. PRUEBAS Y RESULTADOS Se mostrarán los resultados obtenidos en la fase experimental de este proyecto, demostrando que se cumple con lo exigido en el diseño.

7. CONCLUSIONES: Se narrarán las conclusiones obtenidas tras la realización de este proyecto.

8. LÍNEAS FUTURAS: Por último, en este capítulo figurarán los trabajos futuros que se pueden llevar a cabo sobre la base del trabajo realizado.

Capítulo 2

GESTIÓN Y ORGANIZACIÓN DEL PROYECTO

Un proyecto tecnológico se considera exitoso cuando cumple con las especificaciones exigidas por el cliente, satisface a los usuarios, es mantenible (posibilidad de actualización de forma sencilla) y, sobre todo, está desarrollado dentro del presupuesto y del tiempo estimado para ello. Esto implica que por cualquier motivo, como falta de recursos, fallo en los objetivos, obviarse técnicas… el proyecto pueda considerarse como fracaso.

Por ello, todo proyecto necesita de una estructuración en la que se planifique los pasos a realizar, cuánto tiempo se va a emplear en cada paso (planificación), y qué se va a tratar en cada paso (gestión u organización). En este apartado se va a tratar de describir esas diferentes etapas, es decir, cómo se va a gestionar y organizar cada una de las partes más importantes del proyecto.

Desde los años setenta, cuando se empezó a definir qué es considerado Ingeniería del Software [35], uno de los objetivos ha sido la búsqueda y creación de metodologías y estándares que consigan definir la creación del software como una cadena de trabajo continua, de forma que siguiendo esa metodología o estándar, se pueda crear un producto software más fácilmente. Actualmente existen varias metodologías y estándares que permiten crear software de una forma ordenada.

Este proyecto se basa, en concreto, en el estándar diseñado por la Agencia Espacial Europea (ESA) [36] en 1991, cuyo nombre es PSS-05-0 [37]. Este estándar tiene en cuenta la variación que un proyecto software puede tener en tamaño, complejidad, propósito y cantidad de recursos. Dependiendo de estos factores principales, además de otros secundarios como criticidad del software o riesgos y estabilidad de los requerimientos, el estándar distingue dos tipos distintos de proyectos, un proyecto normal, y un proyecto a menor escala (a este último tipo pertenece este proyecto).

La principal diferencia entre los dos tipos de proyectos es la duración, ya que el proyecto es pequeño si este es menor de dos años-hombre; sino el proyecto es un proyecto normal. Otras diferencias menos significativas entre los tipos de proyectos son:

- El equipo de trabajo (menor de 5 personas es un proyecto pequeño).
- El número de líneas de código (menor de 10000 líneas es un proyecto pequeño).

Una vez definido qué estándar se va a seguir durante el proyecto y qué tipo de proyecto se está realizando, se pasa a definir cada una de las características o pasos que se van a aplicar a partir del estándar descrito anteriormente.

2.1 Ciclo de vida del Software

Un producto software se inicia con la definición de los requisitos necesarios para el sistema y finaliza cuando el producto está en desuso y no se le realizan más actualizaciones o mantenimientos. Todo el desarrollo desde la fase inicial hasta la fase final, definiendo cada una de las sub-tareas incluidas en este proceso así como el enlace o comunicación entre las sub-tareas es lo denominado ciclo de vida.

Además de definir las tareas, también comentar que el ciclo de vida asegura que no se produzcan errores en el proceso, ya que su intención es que todos los productos se realicen siguiendo un ciclo de vida definido anteriormente, creando un proceso

"automatizado". Para este proyecto, dentro de los diferentes ciclos de vida existentes, se ha elegido el Modelo en Cascada, en su versión Retroalimentada.

2.1.1 Modelo en cascada retroalimentada

Este modelo está basado en el Modelo en Cascada, también conocido como "Modelo clásico" o "Modelo lineal tradicional", que fue diseñado entre 1966 y 1970 [38].

La idea básica de este modelo consiste en la definición completamente ordenada de cada una de las fases del ciclo de vida, de forma que una etapa no puede comenzar hasta que no ha finalizado la etapa anterior. La idea principal es muy lógica y útil, pero en el universo del desarrollo del software es conocido que es imposible realizar un proyecto sin tener que retocar cualquier cosa de una etapa anterior a la que se encuentra en ese momento el desarrollo.

A pesar de esta idea básica, existen diferentes variantes que desarrollan este modelo, ya que, debido a la simplicidad y eficacia del modelo, es uno de los más utilizados en la creación de software, sobre todo en los proyectos a pequeña y mediana escala. Entre estas variantes existe, por ejemplo, una basada en la refinación de etapas, o aquella que vamos a utilizar en este proyecto, y que está basada en la idea de la retroalimentación.

La retroalimentación consiste en una "ida y vuelta", es decir, que la salida de una etapa puede utilizarse como entrada de una etapa anterior. Por ejemplo, si ya se han definido y especificado los requisitos del sistema, se pasará a la etapa de diseño del sistema, pero lo más seguro es que en esa etapa el equipo de desarrollo se dé cuenta de algunos errores cometidos en la especificación de requisitos y deban volver atrás para realizar los ajustes necesarios.

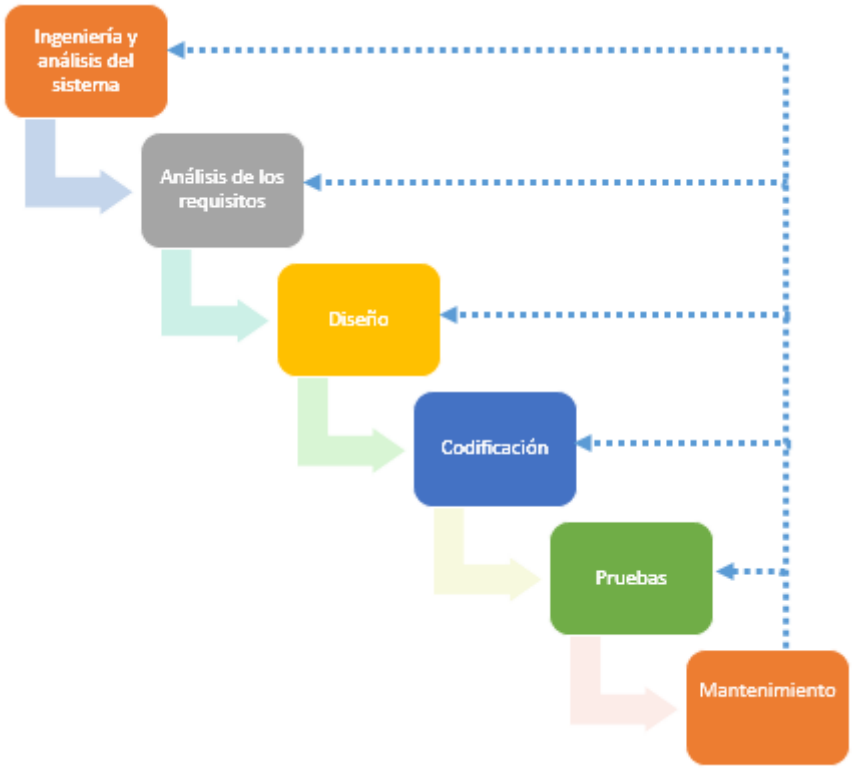


Ilustración 3: Etapas del Modelo en Cascada Retroalimentado

Las distintas etapas con las que cuenta este modelo, y que estas definidas en el diseño que se encuentra en la Ilustración 3, son las siguientes:

- Ingeniería y análisis del sistema: Debido a que el software casi siempre es parte de otro sistema mayor que el propio software, la primera etapa consiste en estudiar el sistema y establecer que requisitos son necesarios en el sistema, para conocer cuáles de esos requisitos pertenecen a nuestro software.

- Análisis de requisitos: En la segunda etapa se extrae que es lo que el cliente demanda y se especifica cuáles son los requisitos a cumplir por el software. Todos estos requisitos serán consensuados sin posibilidad de cambio en el futuro. El documento creado en esta etapa se denomina Documento de Especificación de Requisitos (SRD), que especifica lo que hará el sistema sin detalles de carácter interno.

- Diseño del sistema: En esta etapa se realiza la traducción de los requisitos especificados en el análisis de requisitos a un diseño del sistema, que se divide en dos: el diseño de alto nivel o diseño arquitectónico y el diseño detallado. El diseño desarrollado se basa en cuatro pilares básicos: la arquitectura del software, la estructura de los datos, el detalle procedimental y la personalización de la interfaz. El documento creado en esta etapa se denomina Documento de Diseño del Software (SDD), que especifica la estructura global del sistema y, a su vez, cada parte del software, describiendo también como se deben comunicar cada una de las partes.

- Codificación: Se realiza la traducción del diseño detallado a código fuente legible para el computador. Lo idóneo sería que este paso fuera totalmente automatizado y que, a partir del diseño detallado, se obtuviera el código fuente, pero esto es imposible de conseguir con las herramientas de programación actuales.

- Pruebas: Una vez codificado y ensamblado el producto software, se deben realizar una serie de comprobaciones para chequear que el software es estable, se integra correctamente en el sistema y, por supuesto, cumple con lo especificado por el cliente en las primeras fases.

- Mantenimiento: Esta última fase se realiza una vez ha sido entregado el producto al cliente. Se suelen centrar gran cantidad de recursos en esta fase, ya que en ella se controla que, en caso de que el usuario final encuentre errores, corregirlos. Además, en esta etapa se incluye que el software deba sufrir modificaciones debidos a cambios externos al sistema (por ejemplo, SO o algún periférico nuevo), o que el cliente quiera realizar cualquier tipo de actualización o ampliación del software.

Como cualquier otro, este modelo no es perfecto, y a pesar de ser uno de los más utilizados, también posee varias desventajas, como son:

- En casi todos los proyectos, aunque su idea principal sea seguir un desarrollo lineal, se acaba por realizar más de una iteración, lo que hace que el paradigma del método pierda coherencia.

- Normalmente, el cliente no sabe expresar de manera sencilla y concisa que es lo que realmente desea obtener, por lo que resulta muy difícil extraer todos los requisitos del sistema en las primeras fases y puede que a lo largo de la creación del software surjan incertidumbres difíciles de solucionar (por ello es importante el contrato firmado al final del análisis de requisitos).

- Cualquier error que sea descubierto en la fase de pruebas conlleva al rediseño y nueva codificación, lo que aumenta el coste del desarrollo.

2.2 Requisitos de usuario y software

Como hemos visto en el apartado acerca del ciclo de vida del software, existen varias fases en las que se define qué es lo que el software debe hacer (en base al usuario) y cómo debe interactuar con el sistema (tratado por el ingeniero). Cada uno de los puntos que define alguna característica de qué o cómo debe actuar el producto, es lo que se considera en Ingeniería del Software *requisito*.

Según el estándar en el que este proyecto se basa (PSS-05-0), se distinguen dos tipos de requisitos:

- Requisitos de Usuario: Son aquellos puntos que el cliente demanda al software, aquello que el software debe ser capaz de hacer. Son definidos por el usuario, pero el ingeniero o analista debe ser capaz de, en base a entrevistas abiertas, reuniones en grupo con los usuarios finales, observación del entorno de trabajo, etc., extraer de la manera más exacta y concisa que es lo que el cliente quiere. En este proyecto se van a usar dos tipos de requisitos de usuario: Capacidad (que cosas debe hacer el software) y Restricción (limitaciones del software acerca de cómo debe construirse o debe operar).

- Requisitos de Software: Son aquellos requisitos que definen el comportamiento del sistema a desarrollar. Son definidos por el ingeniero, ya que en base a lo que ha aprendido del cliente con los Requisitos de Usuario, debe conocer que características del software ajenas a la funcionalidad debe poseer el producto.

  En este proyecto se van a usar los siguientes tipos de requisitos de software: Funcionales (aquellos que dicen que debe realizar el software), Rendimiento (especifican el valor de variables medibles como frecuencia…), y No funcionales dentro de los cuales se incluyen: Interfaz (aquello que dicen los elementos del sistema con los que debe interactuar el software), Operacionales (especifican cómo el sistema se comunica con el ser humano y el resto del software), Recursos (definen los requisitos físicos de la maquina donde correrá el software) y Seguridad (precisan cual debe ser la seguridad del software respecto a integridad, disponibilidad…)

A la hora de especificar un requisito, los dos tipos de requisitos cuentan con los mismos atributos, que son los siguientes: identificador, nombre, necesidad, prioridad, estabilidad, fuente, verificabilidad y descripción.

Según esto, un ejemplo de requisito sería el mostrado en la siguiente tabla:

ID: RU-C01			
Nombre:	Nombre		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	Descripción		

Tabla 1: Ejemplo tabla de requisitos

Tal y como se asegura en el estándar seguido, todos los requisitos del documento deben ser verificables, es decir, que se debe usar algún mecanismo para demostrar que todos los requisitos se están implementando a lo largo de todo el proyecto. Para ello, el

mecanismo implementado con los requisitos de usuario y de software es la llamada Matriz de Trazabilidad [39], que chequea que cada uno de los requisitos de usuario esta implementado en, al menos, un requisito de software. Si en algún caso esto no se cumple, se deberían volver a chequear los requisitos, ya que algo en el proceso está fallando.

La definición de los requisitos se llevará a cabo en el capítulo 5 de este proyecto.

2.3 Diseño arquitectónico y detallado.

Siguiendo el ciclo de vida utilizado en este proyecto, una vez extraídos los requisitos que el usuario y el ingeniero consideran necesarios, es el momento de pensar cómo queremos que el software esté construido. Haciendo un símil con la construcción de cualquier edificio, antes de empezar a construir se necesita definir la estructura, funcionamiento e interacción entre las distintas partes del software, lo que es llamado el diseño del software.

Todo diseño del sistema debe seguir los siguientes principios básicos:

- Modularidad: Cada sistema debe estar formado por una jerarquía de módulos. Los módulos de niveles inferiores son menores en alcance y tamaño comparados con los módulos de nivel superior y sirven para fragmentar procesos en funciones separadas.

- Acoplamiento: Los módulos de un sistema deben tener poca dependencia entre sí.

- Cohesión: Los módulos deben llevar a cabo sólo una función de procesamiento.

- Extensión de Control: Los módulos deben interactuar entre si y coordinar las funciones de un número limitado de módulos de nivel inferior.

- Tamaño: El número de instrucciones contenidas en un módulo debe ser limitado.

- Uso compartido: Las funciones no deben repetirse en módulos separados, sino establecerse en un único módulo que pueda utilizar cualquier otro cuando sea necesario.

A la hora de realizar el diseño del software, existen dos etapas claramente diferenciadas dependiendo del nivel de abstracción en el que se encuentre el diseño. En el diseño arquitectónico (también llamado 'diseño de alto nivel'), se intenta llegar a una buena comprensión del problema sin entrar en cómo va a ser la solución en cuanto a detalles de implementación. Una vez realizado esto y justo antes de realizar la codificación del software, se lleva a cabo la segunda etapa del diseño, el diseño detallado, en el que se describe la lógica del programa, el control jerárquico, las estructuras de datos, la unión de componentes, etc.

Al igual que ocurría en la etapa anterior, donde los requisitos de usuario debían ser verificados en los de software, en esta fase también se deberá realizar la verificación de que todo requisito de software debe ser satisfecho por, al menos, un componente de la arquitectura desarrollada, con el fin de asegurar la continuidad del proyecto hasta su etapa de codificación.

El diseño del software se abordará de forma más detallada en el capítulo 5.

2.4 Implementación del software

Una vez concluido el diseño del software, y con toda la información necesaria para llevar a cabo la realización del programa, se lleva a cabo la labor correspondiente a esta etapa, la codificación de la aplicación.

A pesar de que esta es la etapa que cuenta con menor documentación escrita, es a su vez, la más importante y larga del proyecto, ya que todos los pasos anteriores están dirigidos a la correcta creación del software, que es el trabajo a realizar en este punto del proyecto. Además, en este paso se crea aquello que realmente se va a entregar al cliente, ya que toda la documentación escrita no es entregada al cliente porque es útil solo para el equipo de desarrollo.

2.5 Evaluación de los resultados

En este último paso del ciclo de vida de nuestro proyecto, se realizará una evaluación del software, comprobando que funciona correctamente y que los objetivos definidos al principio del proyecto han sido introducidos en el código final.

Para realizar esta evaluación, se utilizará una batería de pruebas del sistema. Esta batería consiste en un conjunto de pequeñas ejecuciones de la aplicación, explorando todas sus posibilidades en busca de errores o inconsistencia de datos, además de chequear que aquellas funcionalidades definidas cuando se creó la arquitectura, realmente han sido implementadas.

Toda la información correspondiente a la evaluación y pruebas del sistema se encuentra en el capítulo 6.

Capítulo 3

ESTADO DEL ARTE

Este capítulo sirve como base teórica sobre la que se sustenta este Proyecto de Fin de Carrera, y en el que se ofrece una visión general del área en el que se enmarca el mismo.

Se comienza hablando sobre el sistema operativo a emplear, desde un punto de vista histórico, su arquitectura y funcionamiento interno, características, diferencias entre versiones y comparación con otros SOs (tanto en características como en mercado). Seguidamente se comenta el protocolo de comunicación empleado, *publish/subscribe* más concretamente la tecnología MQTT. Se prosigue con las herramientas empleadas para la realización del *frontend* y finalmente se comentan aplicaciones existentes similares a la que se persigue en este proyecto.

3.1 Android

3.1.1 Historia de Android

Allá por octubre del año 2003, Andy Rubin, Rich Miner, Nick Sears y Chris White daban forma a Android Inc., la compañía con sede en Palo Alto (California) que comenzó a 'fabricar' las piezas con las que se iría montando, poco a poco, este gigantesco robot verde que hoy disfrutamos.

En sus inicios, únicamente trascendió que la actividad de la empresa se centraba en el desarrollo de software para teléfonos móviles.

En Agosto de 2005 Google adquirió la compañía [4] en una época en la que google aumento su portfolio a base de comprar prometedoras *startup*.

La fecha clave para llegar a entender mejor el éxito de Android es el 5 de noviembre de 2007. Ese día se fundaba la OHA (Open Handset Alliance), una alianza comercial de 35 componentes iniciales liderada por Google, que contaba con fabricantes de terminales móviles, operadores de telecomunicaciones, fabricantes de chips y desarrolladores de software. El mismo día se dio a conocer por vez primera lo que hoy conocemos como Android, una plataforma de código abierto para móviles que se presentaba con la garantía de estar basada en el sistema operativo Linux. Pocos días después, en concreto el 12 del mismo mes de noviembre, se liberaba el primer kit de desarrollo de aplicaciones (sdk) para que los programadores comenzasen a desarrollar las aplicaciones.

El sistema iría madurando hasta que el 23 de Septiembre de 2008 se anunciase el HTC G1, el G1 era un móvil fabricado por HTC el cual incluía la versión 1.0 de Android, y era el primer teléfono comercial que contaba con el sistema operativo Android. [5]

3.1.2 Características

Cada distinta versión de Android que es lanzada dispone de nuevas mejoras y características específicas, pero como base, el sistema operativo fue diseñado para soportar las siguientes:

- Diseño de dispositivo: La plataforma es adaptable a pantallas más grandes, a VGA, a biblioteca de gráficos 2D y 3D, y en general al diseño de móviles tradicionales.

- Almacenamiento: Emplea la base de datos SQLite por ser ligera y con propósitos de almacenamiento de datos.

- Conectividad: Dentro del ámbito de la conectividad, Android soporta las siguientes tecnologías: EV-DO, CDMA, GSM/EDGE, IDEN, UMTS, LTE, Bluetooth, Wi-Fi, HSDPA, HSPA+ y WiMAX.

- Mensajería: Soporta SMS y MMS como formas tradicionales de mensajería de texto, así como C2DM (Android Cloud to Device Messaging), como parte del servicio que ofrece Android de Push Messaging.

- Navegador web: El navegador que se incluye por defecto en Android está basado en WebKit, junto con el motor JavaScript V8 de Google Chrome. En versiones de Android recientes se incluye Google Chrome por defecto.

- Soporte de Java: No existe una máquina virtual de Java en el dispositivo a pesar de que la mayoría de las aplicaciones estén escritas en Java. El soporte para J2ME puede ser añadido mediante aplicaciones de terceros (J2ME MIDP Runner [7]).

- Soporte multimedia: Dentro del ámbito del multimedia Android soporta los siguientes formatos: WebM, H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF y BMP [8].

- Soporte para *streaming*: Android soporta el *Streaming* RTP/RTSP, la descarga progresiva de HTML mediante el tag <video> de HTML5 y el Adobe Flash Streaming (mediante Adobe Flash Player). Se planea el soporte de Microsoft Smooth Streaming con el *plugin* de Silverlight para Android.

- Soporte para hardware adicional: Soporte para pantallas táctiles, GPS, acelerómetros, cámaras de fotos y de vídeo, giroscopios, magnetómetros, termómetro, sensores de proximidad y de presión, y aceleración por GPU (2D y 3D).

- Entorno de desarrollo: El entorno integrado es Eclipse empleando un *plugin* de "Herramientas de Desarrollo de Android", e incluye herramientas para depuración de memoria, un emulador de dispositivos y análisis de rendimiento del software.

- Google Play: Google Play sustituye al anterior Android Market con aplicaciones tanto gratuitas como de pago y que pueden ser obtenidas sin necesidad de un PC.

- Multi-táctil: Android ofrece soporte multi-táctil de forma nativa para pantallas capacitivas.

- Bluetooth: Ofrece soporte para A2DF y AVRCP, así como envío de archivos (OPP), exploración del directorio telefónico, marcado por voz y envío de contactos entre teléfonos.

- Videollamada: A partir de HoneyComb Android soporta videollamadas a través de su aplicación de Google Talk.

- Multitarea: Se dispone de multitarea real de aplicaciones, recibiendo ciclos de reloj las aplicaciones que no se estén ejecutando en primer plano.

- Características basadas en voz: Soporta la búsqueda en Google a través de voz.

- *Tethering*: Permite al dispositivo ser usado como punto de acceso inalámbrico (y alámbrico), aunque para permitir a un PC usar la conexión 3G del dispositivo, podría ser necesaria la instalación de software adicional [9].

3.1.3 Versiones

A continuación las distintas versiones:

Versión	Descripción	Logo
1.0	Fue liberada el 23 de Septiembre de 2008 [10]	
1.1	Fue liberada el 9 de Febrero de 2009 [11]	
1.5	**Android Cupcake** (Basado en el Kernel de Linux 2.6.27) • Lanzamiento: 30 de Abril de 2009 • Funciones: Capacidad de subir videos a YouTube e imágenes a Picasa de forma directa, soporte para Bluetooth (*A2DP* y *AVRCP*), teclado con predicción de texto, transiciones de pantallas animadas, grabar y reproducir videos, y nuevos *widgets*. • Notas: A pesar de que existieran versiones previas de Android, esta marca el inicio del S.O.	
1.6	Android "**Donut**": **(**Basado en el Kernel de Linux 2.6.29**)** • Lanzamiento: 15 de Septiembre de 2009 • Funciones: Se mejora la experiencia en el Android Market, soporte para *CDMA/EVDO*, 802.1x, *VPN* y *WVGA*, mejora velocidad en aplicaciones de búsqueda y cámara, búsqueda por voz y navegación gratuita *turn-by-turn* de *Google*. • Notas: Versión barata que no encarecía el coste de los dispositivos, e inicia de ya de forma importante el auge en la comunidad de desarrolladores.	
2.0 / 2.1	Android "**Eclair**": (Basado en el Kernel de Linux 2.6.29) • Lanzamiento: 3 de Diciembre de 2009 y 12 de Enero de 2010 respectivamente • Funciones: Velocidad hardware optimizada, soporte para *HTML5*, para *Microsoft Exchange*, más resoluciones y para el flash de la cámara. *Bluetooth* 2.1, mejoras en *Google Maps 3.1.2* y en teclado virtual, nuevas listas de contactos,	

IU mejorada, fondos de pantalla animados y zoom digital.

- Notas: Comienza a demostrar su estabilidad y robustez como S.O. para *smartphone*s.

2.2 Android **"Froyo":**
(Basado en el Kernel de Linux 2.6.32)

- Lanzamiento: 20 de Mayo de 2010
- Funciones: Se produce una optimización general sobre memoria, rendimiento y velocidad de aplicaciones; se integra el motor *JavaScript V8* de *Google Chrome*; *Wi-Fi hotspot* y *tethering* por *USB*; permite desactivar el tráfico de datos; marcación por voz; soportes varios (contraseñas, carga de archivos en el *Browser*, instalación de aplicación en memoria expandible, Adobe Flash 10.1, pantallas con alto número de "puntos por pulgada", etc.).
- Notas: Los usuarios aceptan el sistema operativo de forma excelente y comienza a asentarse a nivel mundial.

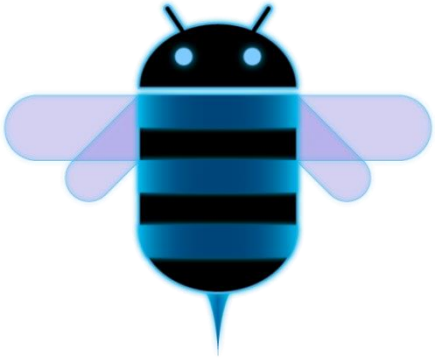
2.3 Android **"Gingerbread":**
(Basado en el Kernel de Linux 2.6.35.7)

- Lanzamiento: 6 de Diciembre de 2010
- Funciones: Soporte para pagos mediante *NFC*, nativo para *VoIP SIP*, múltiples cámaras, para *WebM/VP8* y *AAC*, para más sensores (giroscopio, barómetro); mejoras en batería y administrador de tareas, en entrada de datos (para desarrolladores de juegos), en video online y en el teclado virtual; cortar/copiar/pegar a lo largo del sistema; sistema de archivos *ext4*.
- Notas: Ofrece más posibilidades a usuarios que estén totalmente conectados.

3.0 / 3.1 / 3.2

Android "**Honey Comb**":

- Lanzamiento: 22 de Febrero de 2011
- Funciones: Se realiza una mejora en el sistema multitarea (*multitasking*), escritorio en 3D, mejoras en el navegador web, soporte mediante *Google Talk* para videochat, mejora el soporte para *Wi-Fi*, añade soporte para una gran variedad de periféricos y los *widgets* pueden ser redimensionados (sin la limitación del número de cuadros).
- Notas: Versión optimizada para *tablets*, la versión 2.3 se mantiene para *smartphone*s.



4.0

Android "**Icecream Sandwich**":

- Lanzamiento: 19 de Octubre de 2011
- Funciones: Nueva interfaz y nueva fuente, opción de emplear los botones virtuales, aceleración por hardware, mejora en la multitarea, gestor de tráfico de datos, mejora en el corrector de texto, modificación de notificaciones, mejora de la aplicación de la cámara, *Android Beam* (para compartir contenido entre teléfonos), reconocimiento facial y por voz, soporte nativo para *MKV* y *Stylus*, y nuevo *framework* para aplicaciones.
- Notas: Primera versión que unifica tanto el diseño para *tablets* como para *smartphone*s.

4.1 / 4.2	Android "**Jelly Bean**": • Lanzamiento: 28 de Junio de 2012 • Funciones: Empleo de *Project Butter* para la mejora de la velocidad, optimizando la interfaz para ofrecer mayor rapidez en los efectos y no existan "tirones". Modificación de la barra de notificaciones (más sencilla y limpia) y la introducción de gestos. Nuevo sistema de predicción de texto, dictado de voz *offline*, empleo de voz tanto para búsquedas como para el sistema en sí y *Google Now*. *Google Chrome* como navegador por defecto, fin al soporte de *Flash Player* y cifrado de aplicaciones. • Notas: Última versión hasta la fecha del sistema operativo de Google para *smartphones*/*tablets*.	

Tabla 2: Versiones Android

Las versiones de Android reciben el nombre de postres en inglés. En cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético:

A: Apple Pie (v1.0), Tarta de manzana

B: Banana Bread (v1.1), Pan de plátano

C: Cupcake (v1.5), Magdalena glaseada.

D: Donut (v1.6), Rosquilla.

E: Éclair (v2.0/v2.1), pastel francés conocido en España como pepito,

F: Froyo (v2.2), (Abreviatura de «frozen yogurt») Yogur helado.

G: Gingerbread (v2.3), Pan de jengibre.

H: Honeycomb (v3.0/v3.1/v3.2), Panal de miel.

I: Ice Cream Sandwich (v4.0), Sándwich de helado.

J: Jelly Bean (v4.1/v4.2), Judía de gominola.

K: Key Lime Pie (¿?) Tarta de lima.

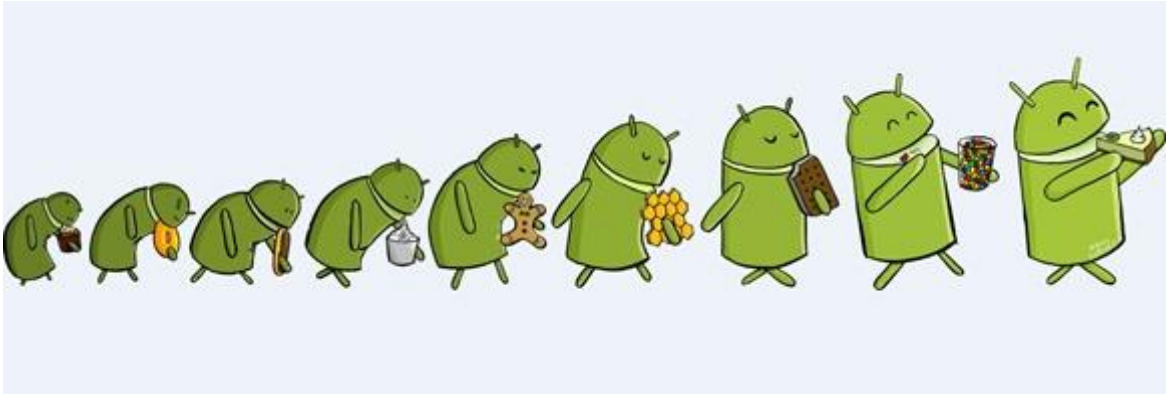


Ilustración 4: Evolución de Android

Unas de los mayores problemas con los que se encuentra la plataforma de Android, es la fragmentación de las versiones. Esto es debido al poco interés mostrado por fabricantes y compañías telefónicas en la actualización de los dispositivos centrándose más en el desarrollo de nuevo modelos en vez de dar soporte y actualizaciones a los modelos ya comercializados, esto sin contar con los modelos de gama baja los cuales ya empiezan su distribución con versiones antiguas del sistema operativo.

A fecha Enero del 2013 la fragmentación estaba distribuida de la siguiente forma

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.4%
2.2	Froyo	8	9.0%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	47.4%
3.1	Honeycomb	12	0.4%
3.2		13	1.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.1%

4.1	Jelly Bean	16	9.0%
4.2		17	1.2%

Tabla 3: Distribución versiones Android

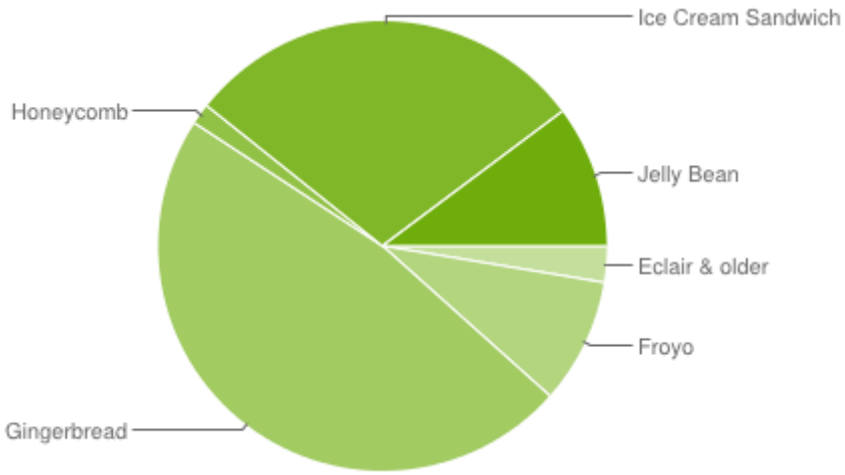


Ilustración 5: Fragmentación Android [12]

3.1.4 Arquitectura

Android está formada por varias capas que facilitan al desarrollador la creación de aplicaciones. Además, esta distribución permite acceder a las capas más bajas mediante el uso de librerías para que así el desarrollador no tenga que programar a bajo nivel las funcionalidades necesarias para que una aplicación haga uso de los componentes de hardware de los teléfonos.

Cada una de las capas utiliza elementos de la capa inferior para realizar sus funciones, es por ello que a este tipo de arquitectura se le conoce también como pila. A continuación se muestra el diagrama de la arquitectura de Android:

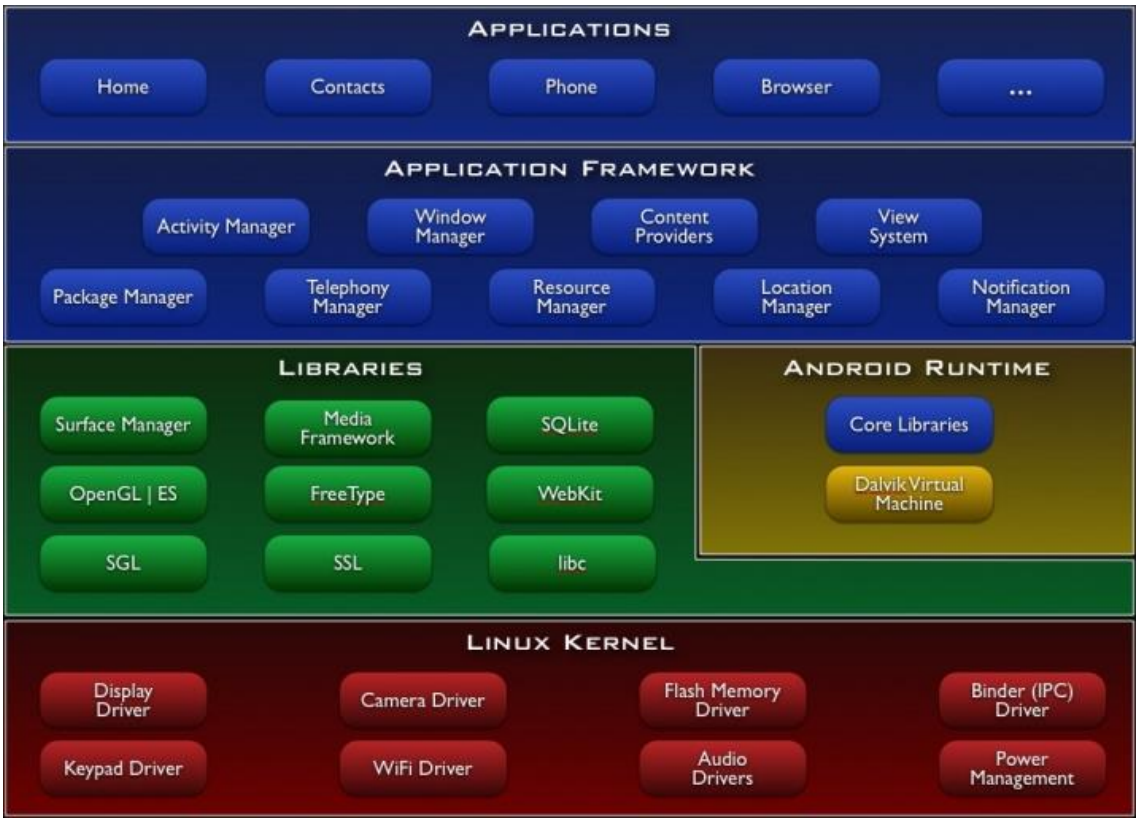


Ilustración 6: Arquitectura Android

Las capas en las que está dividida la arquitectura son las siguientes:

- *Kernel* de Linux: El núcleo del sistema operativo Android está basado en el *kernel* de Linux versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, solo que adaptado a las características del hardware en el que se ejecutará Android, es decir, para dispositivos móviles.

  El núcleo actúa como una capa de abstracción entre el hardware y el resto de las capas de la arquitectura. El desarrollador no accede directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. De esta forma también se evita el hecho de conocer las características precisas de cada teléfono. Si necesitamos hacer uso de la cámara, el sistema operativo se encarga de utilizar la que incluya el teléfono, sea cual sea. Para cada elemento de hardware del teléfono existe un controlador dentro del *kernel* que permite utilizarlo desde el software.

  El *kernel* también se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (*networking*), etc.

- Librerías: La siguiente capa que se sitúa justo sobre el *kernel* la componen las bibliotecas nativas de Android, también llamadas librerías. Están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono. Estas normalmente están hechas por el fabricante, quien también se encarga de instalarlas en el dispositivo antes de ponerlo a la venta. El objetivo de las librerías es proporcionar funcionalidad a las aplicaciones para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se llevan a cabo de la forma "más eficiente".

  Entre las librerías incluidas habitualmente encontramos OpenGL (motor gráfico), Bibliotecas multimedia (formatos de audio, imagen y vídeo), Webkit (navegador), SSL (cifrado de comunicaciones), FreeType (fuentes de texto), SQLite (base de datos), entre otras.

- Entorno de ejecución: Como podemos apreciar en el diagrama, el entorno de ejecución de Android no se considera una capa en sí mismo, dado que también está formado por librerías. Aquí encontramos las librerías con las funcionalidades habituales de Java así como otras específicas de Android.

  El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y de esta forma estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

  Cabe aclarar que Dalvik es una variación de la máquina virtual de Java, por lo que no es compatible con el *bytecode* Java. Java se usa únicamente como lenguaje de programación, y los ejecutables que se generan con el SDK de Android tienen la extensión .dex que es específico para Dalvik, y por ello no podemos correr aplicaciones Java en Android ni viceversa.

- *Framework* de aplicaciones: La siguiente capa está formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones. La mayoría de los componentes de esta capa son librerías Java que acceden a los

recursos de las capas anteriores a través de la máquina virtual Dalvik. Siguiendo el diagrama encontramos:

- Activity Manager: Se encarga de administrar la pila de actividades de nuestra aplicación así como su ciclo de vida.
- Windows Manager: Se encarga de organizar lo que se mostrará en pantalla. Básicamente crea las superficies en la pantalla que posteriormente pasarán a ser ocupadas por las actividades.
- Content Provider: Esta librería es muy interesante porque crea una capa que encapsula los datos que se compartirán entre aplicaciones para tener control sobre cómo se accede a la información.
- Views: En Android, los elementos que nos ayudarán a construir las interfaces de usuario: botones, cuadros de texto, listas y hasta elementos más avanzados como un navegador web o un visor de Google Maps.
- Notification Manager: Engloba los servicios para notificar al usuario cuando algo requiera su atención mostrando alertas en la barra de estado. Un dato importante es que esta biblioteca también permite jugar con sonidos, activar el vibrador o utilizar los LEDs del teléfono en caso de tenerlos.
- Package Manager: Esta biblioteca permite obtener información sobre los paquetes instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes. Con paquete nos referimos a la forma en que se distribuyen las aplicaciones Android, estos contienen el archivo .apk, que a su vez incluyen los archivos .dex con todos los recursos y archivos adicionales que necesite la aplicación, para facilitar su descarga e instalación.
- Telephony Manager: Con esta librería podremos realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso.
- Resource Manager: Con esta librería podremos gestionar todos los elementos que forman parte de la aplicación y que están fuera del código, es decir, cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos o *layouts*.

  - Location Manager: Permite determinar la posición geográfica del dispositivo Android mediante GPS o redes disponibles y trabajar con mapas.
  - Sensor Manager: Nos permite manipular los elementos de hardware del teléfono como el acelerómetro, giroscopio, sensor de luminosidad, sensor de campo magnético, brújula, sensor de presión, sensor de proximidad, sensor de temperatura, etc.
  - Cámara: Con esta librería se puede hacer uso de la(s) cámara(s) del dispositivo para tomar fotografías o para grabar vídeo.
  - Multimedia: Permiten reproducir y visualizar audio, vídeo e imágenes en el dispositivo.

- Aplicaciones: En la última capa se incluyen todas las aplicaciones del dispositivo, tanto las que tienen interfaz de usuario como las que no, las nativas (programadas en C o C++) y las administradas (programadas en Java), las que vienen preinstaladas en el dispositivo y aquellas que el usuario ha instalado.

  En esta capa encontramos también la aplicación principal del sistema: Inicio (*Home*) o lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones mediante una lista y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso *widgets*, que son también aplicaciones de esta capa.

3.1.5 Comparativa con otros SOs

Hoy en día el mercado está monopolizado por los sistemas IOS y Android los cuales abarcan más del 75% de cuota en la mayoría de los mercados, tal y como se puede comprobar en el siguiente tabla

	12 w/e 25 Dec 2011 %	12 w/e 23 Dec 2012 %	Change %
GB	100,0%	100,0%	0,0
iOS	34,1	32,4	-1,7
Android	43,9	54,4	10,5
RIM	16,0	6,4	-9,6
Symbian	3,0	0,7	-2,3
Windows	2,2	5,9	3,7
Bada	0,4	0,1	-0,3
Other	0,4	0,2	-0,2
Germany	100,0%	100,0%	0,0
iOS	23,8	24,7	0,9
Android	60,7	66,6	5,9
RIM	0,9	1,6	0,7
Symbian	8,2	3,2	-5,0
Windows	3,0	2,6	-0,4
Bada	2,7	1,0	-1,7
Other	0,7	0,3	-0,4
France	100,0%	100,0%	0,0
iOS	23,1	25,6	2,5
Android	46,5	58,7	12,2
RIM	12,9	4,7	-8,2
Symbian	3,5	1,0	-2,5
Windows	3,7	4,1	0,4
Bada	9,7	5,3	-4,4
Other	0,7	0,5	-0,2
Italy	100,0%	100,0%	0,0
iOS	20,0	24,9	4,9
Android	50,7	51,8	1,1
RIM	4,8	2,4	-2,4
Symbian	20,1	5,3	-14,8
Windows	2,8	13,9	11,1
Bada	1,2	1,4	0,2
Other	0,5	0,2	-0,3
Spain	100,0%	100,0%	0,0
iOS	8,2	6,4	-1,8
Android	62,2	86,4	24,2
RIM	16,6	2,5	-14,1
Symbian	12,6	2,3	-10,3
Windows	0,4	1,8	1,4
Bada	0,0	0,0	0,0
Other	0,0	0,6	0,6

	12 w/e 25 Dec 2011 %	12 w/e 23 Dec 2012 %	Change %
US	100,0%	100,0%	0,0
iOS	44,9	51,2	6,3
Android	44,8	44,2	-0,6
RIM	6,1	1,1	-5,0
Symbian	0,2	0,1	-0,1
Windows	2,2	2,6	0,4
Bada	0,0	0,0	0,0
Other	1,7	0,9	-0,8
Australia	100,0%	100,0%	0,0
iOS	41,8	38,4	-3,4
Android	48,1	55,8	7,7
RIM	1,1	0,5	-0,6
Symbian	5,9	1,5	-4,4
Windows	1,9	2,8	0,9
Bada	0,0	0,3	0,3
Other	1,2	0,8	-0,4
Urban China	100,0%	100,0%	0,0
iOS	n/a	21,9	
Android	n/a	72,5	
RIM	n/a	0,0	
Symbian	n/a	3,9	
Windows	n/a	0,9	
Bada	n/a	0,0	
Other	n/a	0,9	
Japan	100,0%	100,0%	0,0
iOS	n/a	66,2	
Android	n/a	31,9	
RIM	n/a	0,3	
Symbian	n/a	0,1	
Windows	n/a	0,8	
Bada	n/a	0,0	
Other	n/a	0,8	
EU5	100,0%	100,0%	0,0
iOS	25,4	25,6	0,2
Android	50,8	61,1	10,2
RIM	10,3	4,0	-6,4
Symbian	7,4	2,2	-5,2
Windows	2,6	5,4	2,8
Bada	3,0	1,5	-1,4
Other	0,5	0,3	-0,2

Tabla 4: Cuotas de mercado SOs [13]

En la lucha por el tercer puesto parece que el único que podría competir con Android e IOS es Windows Phone, el cual ha visto aumentada en el último año su cuota de mercado en detrimento de sistemas como Symbian ya prácticamente en desuso y RIM más orientado al entorno empresarial, es por esto por lo que la comparativa va a estar centrada en IOS, Android y Windows Phone.

- Aplicaciones disponibles en el mercado:
  - iOS: Es el sistema que más aplicaciones ofrece a sus usuarios, en torno a 625.000, de las cuales 225.000 están destinadas a iPad.
  - Android: Muy cercano a su competidor, con unas 600.000 aplicaciones.

- Windows Phone: Es el que va más a la cola de todos los sistemas operativos, ofreciendo sólo unas 100.000 aplicaciones a sus clientes.

- Mapas:
  - iOS: Dispone en la actualidad de un sistema propio de mapas, que ofrecen actualizaciones de tráfico, puntos de interés, y descripción de la ruta de navegación. También incluye mapas en 3D. No dispone de ayuda para transporte público.
  - Android: Además del conocido Google Maps, ofrece la posibilidad de ver los edificios en 3D, almacenamiento sin conexión, navegación paso por paso y Street View. También están trabajando sobre el llamado Modo de Brújula, que utiliza un giroscopio del teléfono, para darle vistas interiores de 360 grados.
  - Windows Phone: Incluye mapas de NAVTEQ, la navegación giro a giro, edificios en 3D, almacenamiento en caché sin conexión y enrutamiento dinámico para el transporte público.

- Comandos de voz:
  - iOS: En su anterior versión iOS 5, disponíamos del sistema Siri, que funcionaba bien, aunque no era excelente. Ofrecía dictados de textos, permitía programar eventos en el calendario y establecer contadores de tiempo. En el iOS 6, además se incluirá la capacidad para extraer datos de más fuentes, marcadores deportivos, horarios de película, y reservas en restaurantes. También se está valorando la posibilidad de interactuar con el audio del coche y los diferentes sistemas de navegación.
  - Android: Con Jelly Bean, el reconocimiento de voz se verá mejorado, también reconocerá la entrada de voz, incluso sin conexión.
  - Windows Phone: También dispone de comandos de voz, que permite realizar llamadas, enviar mensajes de texto, buscar en la web y abrir una aplicación desde tu dispositivo, aunque no dispone de la calidad de Apple o Google.

- Funciones de llamada:
    - iOS: iOS 6 permitirá rechazar una llamada, con un SMS de respuesta predeterminado, filtrar las llamadas molestas, e incluirá un conmutador de no molestar.
    - Android: Permite crear una serie de textos que se pueden utilizar como respuestas automáticas rápidas, cuando se rechaza una llamada, y también le permite filtrar las llamadas de personas específicas, pero carece de la capacidad de entrar en un modo No molestar.
    - Windows Phone: Carece de textos pre-integrados ni ningún tipo de No molestar, aunque dispone de filtros avanzados y opciones de bloqueo para las personas que se esté tratando de evitar.

- Mensajería:
    - iOS: IMessage, ofrece muchas promesas, pero aún no hay ninguna certeza de cómo será. Pretende negociar mensajes entre teléfonos, *tablets* y ordenadores portátiles. Actualmente no tienen forma de vincular un número de teléfono y una cuenta de iCloud, lo que impide que los mensajes siempre lleguen, y además, resulta imposible que éstos lleguen a otros usuarios que no tengan sistema iOS.
    - Android: Gracias a GChat los mensajes siempre llegan a todos los dispositivos conectados de manera rápida y fiable. Además, Google Voice está bien integrado en Android, lo que permite que los mensajes enviados desde el ordenador o el teléfono estén bien sincronizados.
    - Windows Phone: Permite enviar sin problemas los mensajes de texto, mensajes de Facebook y de Skype a un determinado contacto desde la misma ventana.

- Integración en Facebook:
    - iOS: Gracias a iOS 6 puedes actualizar tu estado y subir imágenes desde diferentes aplicaciones y también desde el centro de

notificaciones, al igual que la sincronización de contactos y permite coordinar tus eventos de Facebook con los de tu calendario de iOS.

- Android: Puedes compartir y subir cosas desde prácticamente cualquier lugar del OS, o desde la mayoría de las aplicaciones para Android. Además puedes extraer información de Facebook para almacenarla en tus contactos o descargar todos tus contactos de Facebook.
- Windows Phone: Integra perfectamente características como, actualizaciones de estado, imágenes, contactos, chat y eventos en secciones propias de Microsoft.

- Video Chat:
  - iOS: Dispone de FaceTime, que puede realizar llamadas a través de 3G o Wi-Fi, y funciona bastante bien, pero sólo lo hace con otros terminales Apple.
  - Android: Gracias a Gmail / Google Talk se puede chatear por vídeo con cualquier persona que tenga Gmail en un Mac, PC o Android, a través de 3G o Wi-Fi. Se espera una actualización del mismo, con soporte mejorado Hangout, que mejorará el servicio.
  - Windows Phone: Permite el chat de voz de Skype para Mac, PC, IOS y Android.

- Pagos con móvil:
  - iOS: El iPhone no cuenta con NFC, pero dispondrá de PassBook, basadas en GPS y Códigos QR, para tarjetas de crédito/débito, tarjetas de recompensas, incluso proporcionará información sobre actualizaciones u ofertas recientes sobre temas de interés.
  - Android: El sistema Google Wallet incluye pagos móviles, ofertas, regalos, y se mantendrá sin cambios.
  - Windows Phone: Con la llegada al mercado del Windows Phone 8, también llegará el sistema Wallet, que es una especie de cartera digital. Será capaz de almacenar tarjetas de crédito y de débito, tarjetas de regalo y tarjetas de cliente, además de la posibilidad de

acceder a ofertas. Y lo que la diferencia de sus competidores será que hará uso de elementos de seguridad NFC almacenados en tarjetas SIM, lo que permitirá mayor flexibilidad y seguridad en los pagos.

- Iconos inteligentes:
    - iOS: En este sentido Apple no dispone de grandes elementos, casi todo se reduce a notificaciones y alertas para saber cuándo tienes nuevos mensajes, correos electrónicos o notificaciones, pero prácticamente nada más.
    - Android: Cuenta con *widgets* para asumir la tarea de actualizaciones en tiempo real, lo que permite una buena personalización a la hora de ver las actualizaciones de tu correo, calendario o estado del tiempo, desde la pantalla principal, de una manera muy rápida.
    - Windows Phone: Los mosaicos de Windows Phone 8, son los mejores en este sentido. No solo pueden mostrar notificaciones e información relevante, como mensajes de texto, correos electrónicos, etc, sino que también, están organizados en una red muy organizada que estará reforzada por la capacidad de dividir los mosaicos en tres tamaños diferentes, dependiendo de cómo desees que se muestre la información.

- Media Streaming:
    - iOS: Dispone de AirPlay, que permite pasar música de tu ordenador o tu dispositivo iOS a altavoces con AirPlay, *routers* AirPort Express y Apple TV.
    - Android: Con la introducción del Nexus Q, podrás compartir audio y video, pudiendo transmitirlo a televisores u otros sistemas.
    - Windows Phone: Tendrá SmartGlass, que servirá como plataforma de *streaming* a la Xbox, y permitirá simplificar y visualizar el proceso de descargar entre la Xbox y los dispositivos Windows Phone 8. Además, podrás crear

contenido adicional a tu dispositivo mientras ves un programa de TV.

Tabla resumen de la comparativa:



	iOS	Android	Windows Phone
Sistema Operativo	IOS 6	Jelly Bean	Windows Phone 8
Mapas, Street View	NO	SI	NO
Pago con móvil	Passport , no NFC	Google Wallet	Wallet, mayor seguridad
Comando de Voz dictado	Buen nivel reconocimiento por voz	Mejoras importantes en el reconocimiento por voz, es el mejor	Muy simple
Reconocimiento de voz fuera de linea	NO	SI	NO
Video Chat	FaceTime	Gmail / Google Talk	Skype
Funciones para las llamadas	Conmutador de no molestar	No incluye el conmutador de no molestar	No dispone de conmutador de no molestar ni de textos predictivos
Integración con redes sociales	Con Facebook	Con facebook + Integrar y fusionar nuestros amigos en la agenda de contactos + Sincronizar calendarios	Con facebook + Integrar y fusionar nuestros amigos en la agenda de contactos + Sincronizar calendarios
Aplicaciones	625.000	600.000	100.000
Widgets	Si, en centro de notificación	Si	No
Mensajería	IMessage	Gchat	Enviar mensajes de texto, mensajes de Facebook y de Skype a un determinado contacto desde la misma ventana
Iconos Inteligentes	Escasos	Bastantes y muy buenos	Los mosaicos de Windows Phone 8, son lo mejores
Media Streaming	AirPlay	Nexus Q	SmartGlass

Tabla 5: Comparativa SOs móviles.

3.2 Comunicación

Es fácil encontrar aplicaciones móviles con la necesidad de actualizar los datos en tiempo real, que la actualización se realice justo cuando ocurre un evento específico.

Desde hace mucho tiempo se optó por una solución más directa, llamada *polling*. En el caso del *Polling* la aplicación interroga periódicamente al servidor, acosándolo

continuamente, para saber si existen nuevos datos disponibles, lo que hace que sea un método algo ineficiente, gastando muchos recursos de energía y red [15]. Esto no es lo idóneo más si cabe teniendo en cuanta las limitaciones de autonomía de los que disponen actualmente los dispositivos móviles.

La alternativa es un método más avanzado y eficiente: el *server push* (empuje de información por parte del servidor), de esta forma el proceso de envío de información se automatiza, sin necesidad de que lo hubiera solicitado previamente el cliente. Los servicios *push* se basan normalmente en preferencias de información prestablecidas, o lo que también se conoce como paradigma *publish/subscribe*. Dicho paradigma puede ofrecer un alto grado de desacoplamiento entre los elementos de una aplicación distribuida. Es por esta razón por la que *publish/subscribe* es considerada muy adecuada para entornos dinámicos como el móvil [16], donde el conjunto de componentes se somete a continuas reconfiguraciones. En años recientes, varios prototipos han sido presentados demostrando la integración del mecanismo de enrutamiento *publish/subscribe* tanto con redes de sensores [17] como con plataformas móviles [18].

El patrón *publish/subscribe* se trata de una arquitectura de paso de mensajes desacoplada y en algunas implementaciones distribuidas donde existen "publicadores" que envían mensajes ante el acaecimiento de un suceso específico sin conocimiento alguno sobre que va pasar después con el mensaje. Análogamente existen "suscriptores" que en cuya inicialización se define a qué tipo de mensajes se suscribirán (el tipo se define en función del "asunto" o "*topic*") y qué acción debe ejecutarse cuando el mensaje llegue [19].

El patrón *publish/subscribe* facilita el desacople de componentes (módulos, paquetes) dentro de una aplicación. Los conceptos involucrados son:

- Permitir que partes de un aplicación envíen mensajes "al resto de la aplicación" sin tener que conocer:
    - *si* el mensaje será manejado y usufructuado:
        - puede suceder que el mensaje se ignore completamente
        - que sea manejado en muchas partes diferentes de la aplicación
    - *cómo* será manejado el mensaje:

  - al publicador no le importa qué se hará con el mensaje y su contenido.
  - tampoco hay control del orden en que un mensaje dado se enviará al resto de la aplicación.
- Permitiendo que partes de una aplicación reciban y manejen mensajes desde "el resto de la aplicación" sin tener que conocer *quién* envió el mensaje.

Un receptor es una parte de la aplicación que quiere recibir mensajes. Un receptor se suscribe a uno o más tópicos. Un emisor es cualquier parte de la aplicación (deposita en el intermediario) que envía un mensaje con un *tópic* dado, y opcionalmente, cualquier información adjunta. Este intermediario (conocido como *bróker*) entrega este mensaje a todos los receptores suscriptos. [20]

Las ventajas del patrón *publish/subscribe* son:

- Acoplamiento débil: la topología de *publish/subscribe*, basada en la intermediación y el desconocimiento de identidades y comportamientos de los objetos que interactúan permite un desacople de los componentes de la aplicación. Esto significa que las distintas partes de la aplicación son independientes entre sí, de modo que pueden fácilmente desactivarse componentes no críticos sin afectar al conjunto de la aplicación. Esta estrategia es útil para realizar pruebas de seguridad.

- Funcionalidad configurable: Dado que un emisor no tiene necesidad de conocer la existencia de un receptor, es fácil diseñar una arquitectura basada en "*plugins*" que permite mantener un núcleo y agregar funcionalidades extra con posterioridad (incluso desarrolladas por terceros). Esto trae aparejada la posibilidad de adaptar, mediante extensiones que se activan o no, las características del software en función de las necesidades del usuario.

- Escalabilidad: En las implementaciones distribuidas de demanda moderada (donde los mensajes se transmiten entre múltiples procesos o, incluso, equipos), *publish/subscribe* provee una arquitectura mucho más simple y autogestionada que la típica topología cliente/servidor para tareas de procesamiento paralelo.

Existen tres tipos principales de patrones pub/sub que pueden ser usados para crear un mecanismo para enviar los datos: "Basado en el tema" (*Topic-Based*), "Basado en la transmisión" (*Broadcast-Based*), y "Basado en el contenido" (*Content-Based*):

- *Topic-Based*: Los elementos del sistema intercambian información a través de un conjunto de listas de *topic*s predefinidas o nombres de canales lógicos, normalmente conocidos de antemano. Todos los suscriptores de un *topic* recibirán todos los mensajes publicados en dicho *topic*.

- *Broadcast-Based*: Todo mensaje se transmite a todo el sistema. Cada una de las entidades que reciba la información deberá inspeccionarla y en el caso de que esté suscrita al tema del mensaje, procesarla.

- *Content-Based*: En este enfoque, las suscripciones están directamente relacionadas con contenido específico de información, ofreciendo así el modelo pub/sub más flexible y versátil.

A continuación se muestra el diseño de un patrón basado en *topic*

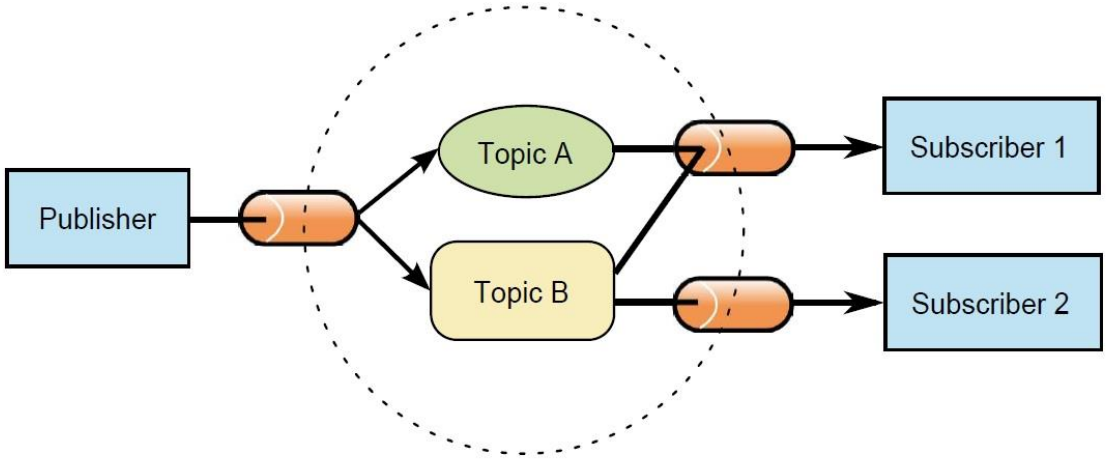


Ilustración 7: Diseño patrón *publish/subscribe topic*-Base

3.2.1 C2DM

Google proporciona un servicio nativo completamente integrado con la plataforma Android, *Cloud to Device Messaging* (C2DM). C2DM es un servicio que ayuda a los desarrolladores a enviar datos desde los servidores a sus aplicaciones de Android.

El servicio proporciona un mecanismo sencillo y ligero para que los servidores pueden contactar con las aplicaciones móviles en busca de una actualización o algún dato de usuario. El servicio C2DM maneja todos los aspectos de la cola de mensajes y la entrega a la aplicación de destino que se ejecuta en el dispositivo de destino. [21]

Los componentes de este Sistema son

- Dispositivo Móvil: El dispositivo en el que se ejecuta la aplicación que usa C2DM. Este debe de ser un dispositivo Android 2.2 o superior, tener Google Play instalado y estar logado al menos con una cuenta Google.

- Servidor de datos: Servidor encargado de gestionar el estado de los datos y comunicar el estado de los mismos. El servidor de datos envía los mensajes a la aplicación Android en el dispositivo a través del servidor de C2DM.

- Servidor C2DM: Servidor de Google encargado de recibir los mensaje del servidor de dato y enviarlos al dispositivo móvil.

A continuación se muestra un diagrama de cómo sería el proceso de registro de una aplicación

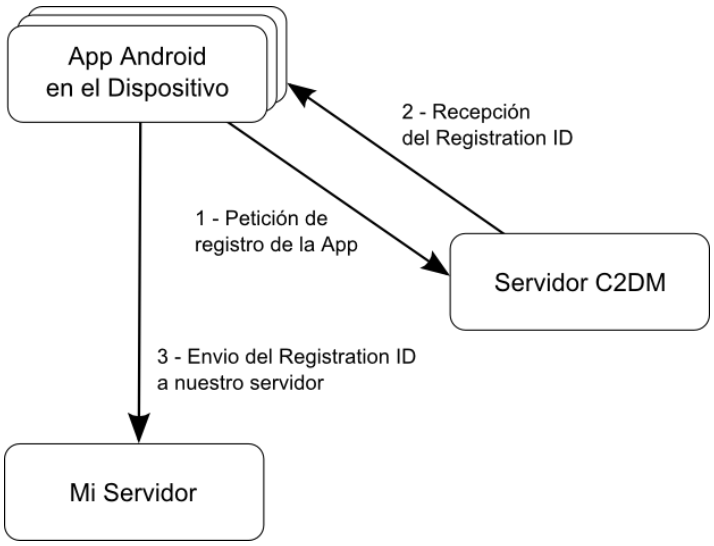


Ilustración 8: Registro aplicación C2DM

Y un diagrama del flujo en el envío de un mensaje

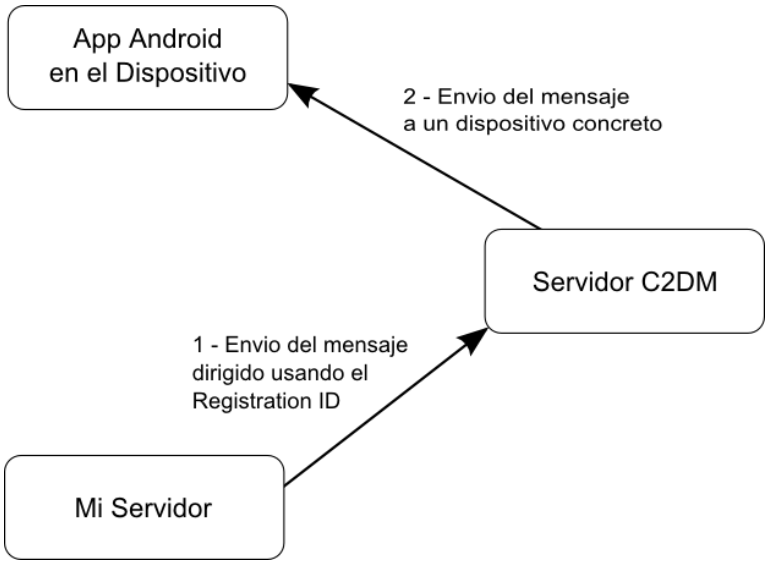


Ilustración 9: Envio mensaje C2DM

3.2.2 MQTT

MQTT es un *broker* ligero basado en el patrón de *publish/subscribe*, es un protocolo de mensajería diseñado para ser abierto, simple, ligero y fácil de implementar. [22] Estas características la hacen ideal para su uso en entornos restringidos, por ejemplo:

- Cuando la red es cara, tiene poco ancho de banda o no es fiable.
- Cuando se ejecuta en un dispositivo embebido con limitada capacidad de procesamiento o de recursos de memoria

Las Características del protocolo son:

- El patrón *publish/subscribe* proporciona la distribución de mensajes de uno a muchos y el desacoplamiento de las aplicaciones.
- El uso de TCP / IP para proporcionar conectividad de red básica
- Tres calidades de servicio para la entrega de mensajes:
  - "A lo sumo una vez", donde los mensajes se entregan de acuerdo a los mejores esfuerzos. Pérdida de mensajes o duplicación de mensajes puede ocurrir. Este nivel podría ser utilizado, por ejemplo, con los datos del sensor ambiente en el que no importa si una lectura individual se pierde ya un nuevo valor se publicará poco después.
  - "Por lo menos una vez", donde los mensajes se aseguran que van a llegar, pero puede ocurrir algún duplicado.
  - "Exactamente una vez", donde el mensaje se asegura que llegara exactamente una vez. Este nivel se podría utilizar, por ejemplo, con sistemas de facturación donde los mensajes duplicados o perdidos podrían producir cargos incorrectos.

- Una sobrecarga de red pequeña (la cabecera de longitud fija está a sólo 2 bytes), y los intercambios de protocolo minimizados para reducir el tráfico de red.

- Un mecanismo para notificar a las partes interesadas a una desconexión anormal de un cliente

El diagrama del diseño con MQTT es el siguiente:

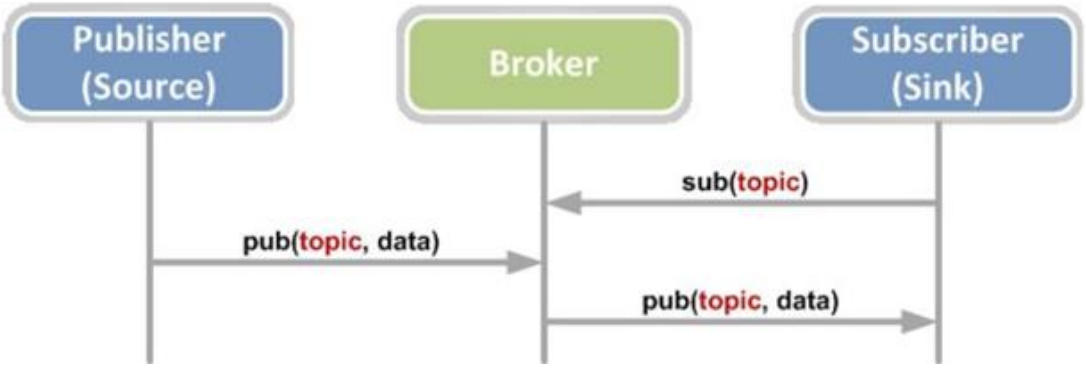


Ilustración 10: Diagrama MQTT

3.3 Herramientas

En este apartado se da una breve introducción a las herramientas utilizadas para la realización de este Proyecto Fin de Carrera, además de resumir las características más importantes de cada una de ellas.

3.3.1 SDK Android

El Android SDK (*Software Development Kit*), descargable desde la página oficial de Android, comprende una serie de herramientas de desarrollo necesarias para crear aplicaciones para este sistema operativo. Dicho SDK está compuesto por paquetes modulares que se pueden descargar de forma separada usando el Android SDK Manager. Algunos de los paquetes disponibles más remarcables son los siguientes [24]:

- SDK Tools: Contiene herramientas de depuración y de pruebas, junto con otras utilidades requeridas para desarrollar una aplicación. Conviene mantenerlo actualizado (al instalar el SDK *starter package* se obtiene la última versión).

- SDK Platform–tools: Contiene herramientas de desarrollo y depuración dependientes de la plataforma, que soportan las últimas características de la plataforma Android y que son típicamente actualizadas cuando una nueva plataforma entra a estar disponible. Estas herramientas son compatibles con versiones de plataformas más antiguas.

- Documentación: Una copia offline de la última documentación de las APIs de la plataforma Android.

- Ejemplos para el SDK: Una colección de aplicaciones de ejemplo basándose en las APIs de la plataforma. Ejemplos pensados sobre todo para desarrolladores iniciados en código de Android.

- Google Play Billing: Proporciona las librerías estáticas y los ejemplos que permiten integrar los servicios de facturación en la aplicación.

- Google Play Licensing: Proporciona las librerías estáticas y los ejemplos para llevar a cabo la verificación de la licencia de la aplicación a la hora de distribuirla.

Hay algunos casos en los que un paquete del SDK pueda requerir una versión específica mínima de otro paquete o del SDK Tools. Por ejemplo podría haber una dependencia entre el *Plugin* ADT para Eclipse y el paquete SDK Tools (por ejemplo, ADT 8.x requiere SDK Tools r8).

Comentar además que para que la tarea de desarrollo sea lo más sencilla posible, se recomienda instalar el driver USB (Windows) [25] y el *plugin* para Eclipse [26], incluido en el kit de desarrollo, el cual integra las características del SDK dentro del entorno del mismo IDE.

3.3.2 Eclipse

Eclipse es un entorno integrado de desarrollo (IDE) multiplataforma para crear aplicaciones clientes de cualquier tipo. Es libre y fue creado originalmente por IBM,

aunque ahora lo desarrolla la Fundación Eclipse, una organización independiente sin ánimo de lucro que apoya una comunidad de código abierto. La aplicación más importante que ha sido realizada con este entorno es el afamado entorno de desarrollo integrado Java, llamado Java Development Toolkit (JDT). Este es el entorno de desarrollo que se está convirtiendo en el estándar de facto para Java, de hecho otros IDE's comerciales como JBuilder han anunciado que su próxima versión se basará en Eclipse [23]

Eclipse no es tan sólo un IDE, se trata de un *framework* ampliable mediante módulos (*plugin*). Estos módulos o *plugins* proporcionan más funcionalidades, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Incluso hay *plugins* que permiten que el entorno de desarrollo soporte otros lenguajes además de Java, como por ejemplo PHP, Perl, etc.

Los componentes gráficos de Eclipse están basados en un juego de herramientas de tercera generación para Java de IBM llamado SWT, que mejora los de primera y segunda generación de Sun. La interfaz de usuario de Eclipse cuenta con una capa intermedia de interfaz gráfica (GUI) llamada JFace, lo que simplifica la creación de aplicaciones basadas en SWT.

3.3.3 MySql

Mysql es la base de datos de código abierto más popular en el mundo [27]. MySQL es un gestor de base de datos completamente desarrollado en lenguaje C/C++, lo que ofrece una estabilidad de trabajo impresionante. También, cuenta entre sus características, con una excelente capacidad de integración con diferentes entornos de desarrollo de software y de aplicaciones cliente/servidor, por lo tanto, es muy popular entre los programadores de aplicaciones web y entre administradores de base de datos en todo el mundo.

Una de las características más atractivas de MySQL, es que cuenta con un sistema de trabajo bastante simple que puede integrarse con apartados de usuario visuales e incluso, permite trabajar en diferentes sistemas informáticos, ya que es una aplicación multiplataforma.

Aunque en la actualidad existe una enorme cantidad de opciones para gestionar bases de datos mucho más avanzadas y completas que MySQL, esta última cuenta con una gran cantidad de opciones de adaptación a diferentes programas de programación web, específicamente PHP, lo que permite crear aplicaciones cliente/servidor de gran calidad y estabilidad, siendo una dupla inseparable para este tipo de desarrollos.

Otra característica destacable de MySQL, que la hace destacar sobre otras opciones, es la facilidad de los comandos de trabajo, lo que la convierte en una herramienta fácil de dominar y de utilizar, requiriendo poco tiempo de capacitación y estudio para realizar las tareas de desarrollo de sistemas de base de datos.

MySQL es un gestor de base de datos bastante versátil, que cuenta con una gran cantidad de opciones de trabajo:

- Cuenta con la capacidad de realizar tareas multiprocesador, debido a que posee la opción de trabajo multihilo.

- Puede ingresar una enorme cantidad de datos por columna de trabajo.

- Cuenta con API's disponibles para los principales lenguajes de programación que existen.

- Aplicación con una portabilidad sobresaliente.

- Capacidad de soportar hasta 32 índices de tablas diferentes.

- Estupendo nivel de seguridad que permite gestionar varios usuarios con *login* y contraseñas individuales.

Su mantenimiento es bastante simple y rápido por lo que se considera a MySQL un gestor de base de datos ideal para desarrollar trabajos web y para organizar aplicaciones locales, donde los sistemas de base de datos sean simples.

3.3.4 Java

Java es un lenguaje de programación y la primera plataforma informática creada por Sun Microsystems en 1995. Es la tecnología subyacente que permite el uso de programas punteros, como herramientas, juegos y aplicaciones de negocios. Java se ejecuta en más de 850 millones de ordenadores personales de todo el mundo y en miles de millones de dispositivos, como dispositivos móviles y aparatos de televisión. [28]

Java fue diseñado para ser:

- Sencillo, orientado a objetos y familiar: Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje.

- Robusto y seguro: Robusto, simplificando la gestión de memoria y eliminando las complejidades de la gestión explicita de punteros y aritmética de punteros del C. Seguro para que pueda operar en un entorno de red.

- Independiente de la arquitectura y portable: Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera *'bytecodes'*, un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software. Es además portable en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El *'bytecode'* es traducido a código máquina y ejecutado por la Java Virtual Machine, que es la implementación Java para cada plataforma hardware-software concreta.

- Alto rendimiento: A pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.

- Interpretado, multi-hilo y dinámico: El intérprete Java puede ejecutar *bytecodes* en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

3.4 Aplicaciones comerciales

3.4.1 Aplicaciones comerciales para la monitorización de sensores

Actualmente existen muchas aplicaciones destinadas a dispositivos móviles, ya sea para Android, IOS, Windows Phone, etc… orientadas a la telemonitorización. A continuación se muestran algunas de estas aplicaciones

3.4.1.1 Sensordrone

Sensordrone es un dispositivo el cual se encuentra lleno de sensores, el dispositivo puede servir para muchas cosas permitiendo medir gases, temperaturas, humedad, radiación solar, etc. Básicamente el dispositivo se conecta con un *smartphone* y una aplicación que te ofrece la información sobre una determinada característica a medir. [29]

Ilustración 11: sensordrone

3.4.1.2 iMon Android

iMon Android es un aplicación gratuita para visualizar imágenes en vivo de sus instalaciones de CCTV en cualquier momento, desde cualquier lugar, a través de un dispositivo móvil basado en Android. Además de visualizar imágenes permite recibir información de eventos y configurar remotamente todos los parámetros de las grabaciones [30]

Ilustración 12: iMon Android

3.4.1.3 mHealthAlert

mHealthAlert es una plataforma de gestión remota de telemedicina, albergada en la nube, que permite la tele-monitorización de personas mayores y enfermos crónicos desde sus domicilios y que da acceso a los médicos a la información clínica del paciente para su inmediato tratamiento. [31]

Ilustración 13: Ciclo mHealthAlert

Ilustración 14: Interfaz gráfica mHealthAlert

3.4.1.4 Glucómetro

Glucómetro es una aplicación de Android para controlar su diabetes, el seguimiento de su nivel de glucosa con un medidor de glucosa bluetooth. [32]

- Seguimiento de HbA1C, comida, el peso, ejercicio, medicación, la presión arterial, pulso.
- Informe Estadístico
- Gráfico
- Calculadora insulina incluida.
- *Widget*
- Conectividad Bluetooth

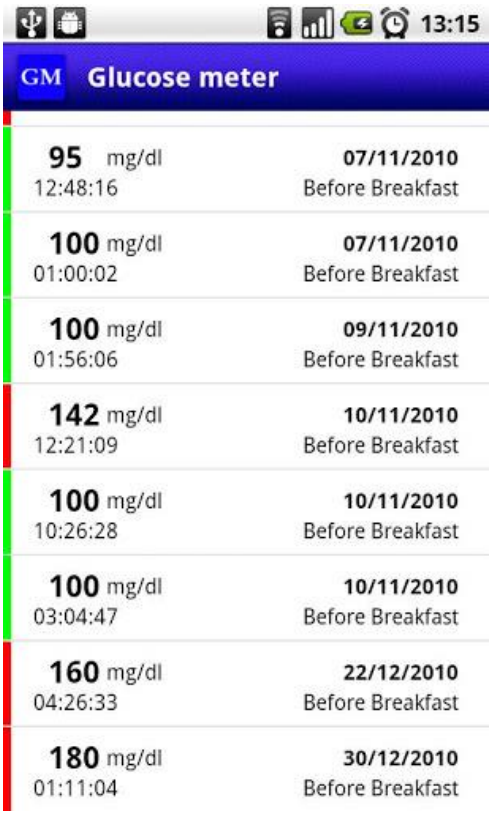


Ilustración 15: Glucómetro

3.4.1.5 Waspmote

Waspmote es una plataforma de código abierto de sensores inalámbricos especialmente centrados en la implementación de modos de bajo consumo para permitir a los nodos de los sensores ser completamente autónomos.

Waspmote es un nodo sensorial inalámbrico con uno de los consumos más bajos (0,7uA) en comparación con cualquier otra plataforma de sensores. Es capaz de medir

más de 50 sensores (desde sensores de gases a agrícolas) y transmitir los datos a través de diferentes protocolos inalámbricos como ZigBee, GPRS/3G, WiFi o RFID/NFC. [33]

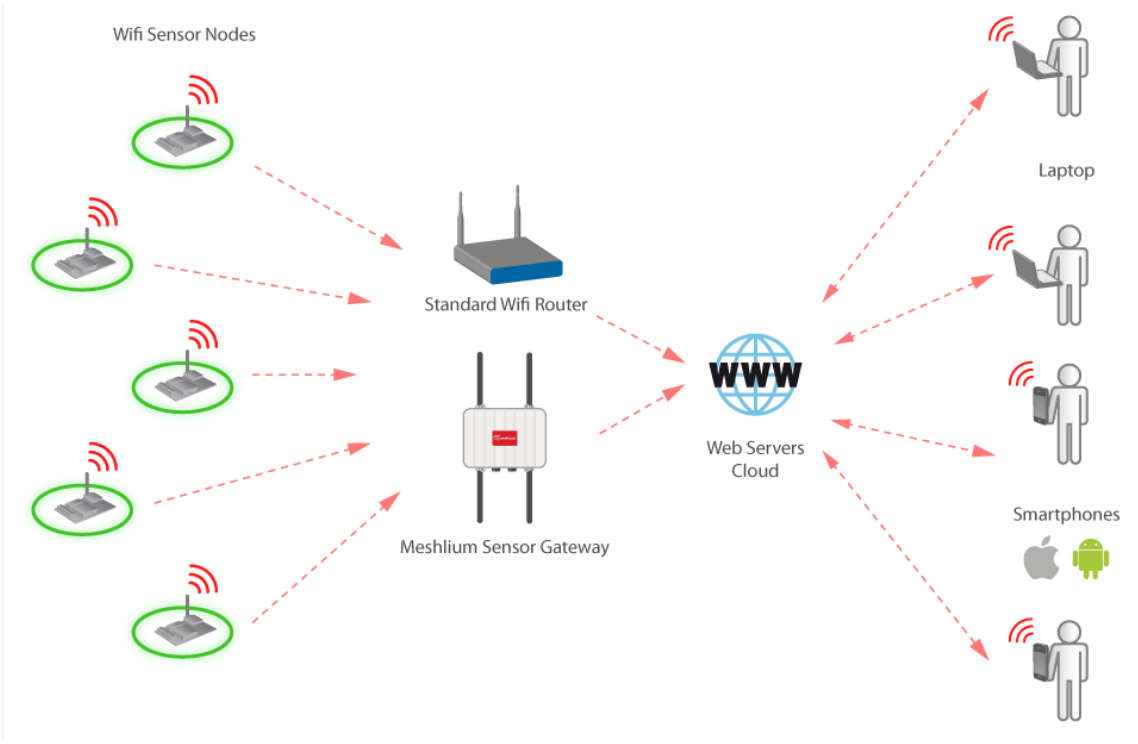


Ilustración 16: Red de sensores Waspmote

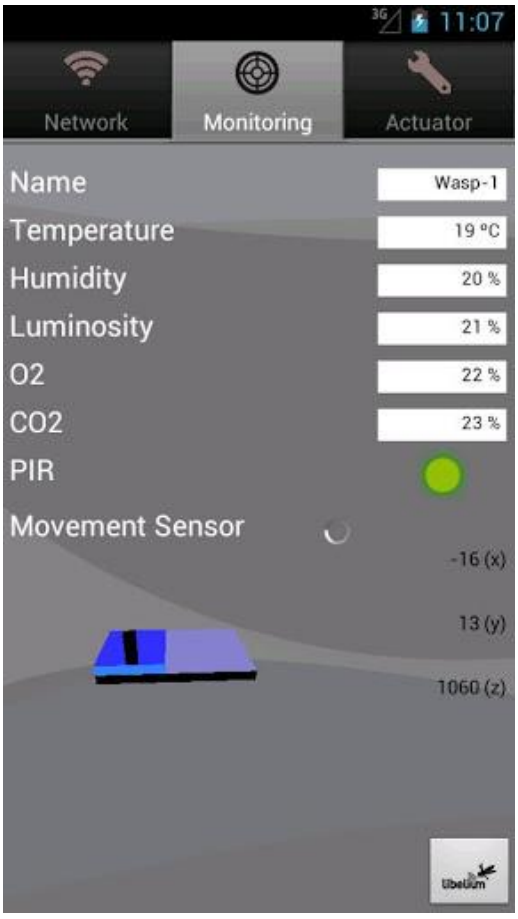


Ilustración 17: Interfaz gráfica Android

3.4.2 Aplicaciones comerciales basadas en tecnología push

Además de las aplicaciones orientadas a la telemonitorización existen otras aplicaciones en las cuales la comunicación en tiempo real es importante. A continuación se muestran algunas aplicaciones que usan la tecnología *push*

3.4.2.1 Gmail

Gmail se basa en la idea de hacer que el correo electrónico sea más intuitivo, eficiente, útil e incluso divertido. Recibe tu correo al instante con notificaciones *push*, consulta tus conversaciones, responde mensajes online y sin conexión, y encuentra cualquier correo. [34]

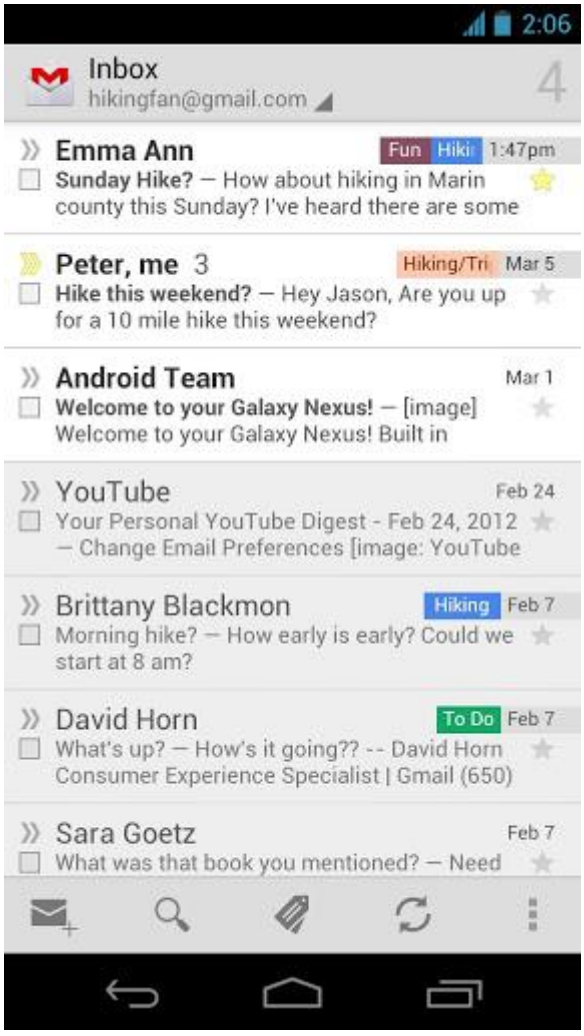


Ilustración 18: GMail

3.4.2.2 Whatsapp Messenger

WhatsApp Messenger es un mensajero multiplataforma disponible en Android en otros sistemas operativos móviles. La aplicación utiliza la conexión 3G o WiFi para recibir mensajes de amigos y familia. Envía mensajes, fotos, notas de audio y videos. Con las notificaciones *push* WhatsApp está siempre activo y siempre conectado.

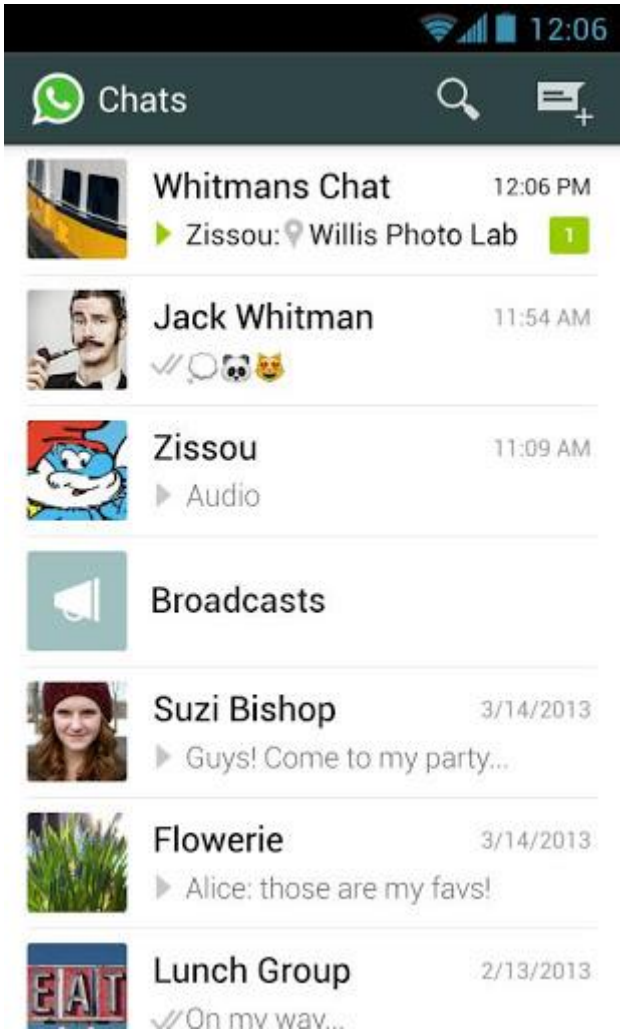


Ilustración 19: Whatsapp

Capítulo 4

OBJETIVOS

Es en este capítulo en el que se presenta la meta y los objetivos principales a perseguir en este Proyecto, enfocado en el ámbito de la telemonitorización mediante Android.

El objetivo general de este Proyecto Fin de Carrera es el diseño e implementación de un sistema de comunicación en tiempo real mediante tecnología *push* para el sistema operativo Android, aplicado a la monitorización de un entorno doméstico

Además de dicho objetivo principal, existen una serie de objetivos específicos a abordar:

- Estudio general de las distintas posibilidades de comunicación de un *Smartphone* en tiempo real.
- Estudio del funcionamiento del bróker MQTT y su parametrización
- Estudio de aplicaciones dirigidas a *smartphone* con relación en el ámbito de la telemonitorización/vigilancia.
- Diseño e implementación de un sistema *backend* que simule una red de sensores y que se encargue de publicar los datos de los sensores en el bróker

MQTT

- Utilización de un protocolo de comunicación para el envío de mensajes y eventos entre el *smartphone* y el *backend.*

- Desarrollo de una aplicación para *Smartphone* para el sistema operativo Android usando el SDK de Android junto con el IDE Eclipse.

- Verificación de que dicha aplicación desarrollada funcione correctamente y su posterior despliegue al ámbito real, es decir, a un *smartphone*.

.

Capítulo 5

DISEÑO E IMPLEMENTACION DE LA APLICACIÓN

A continuación se especifica todo lo relacionado con el desarrollo de la aplicación, comenzando con el detalle de los requisitos establecidos por el usuario (con los consecuentes requisitos software), seguido de los casos de uso y de la arquitectura, concluyendo con la implementación final.

5.1 Análisis

Esta fase se considera sumamente importante en el proceso de desarrollo de un sistema interactivo, al servir de base para comprender y concretar el sistema, y conocer de esta forma los elementos que conforman el contexto del problema. Los productos del análisis a desarrollar serán los siguientes: Requisitos de la aplicación (RUs y RSs) y casos de uso.

5.1.1 Requisitos usuario

En este grupo de requisitos se encuentran los definidos por el usuario, es decir, aquellas características demandadas por el cliente que tienen como objetivo plantear el problema y delimitar qué es lo que ha de cumplir la aplicación.

Los requisitos de capacidad son:

ID: RU-C01			
Nombre:	Estado sensores		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	La aplicación deberá mostrar el estado de los sensores según estos se vayan activándose, mostrándose el estado en una lista de sensores.		

Tabla 6: RU-C01

ID: RU-C02			
Nombre:	Lista sensores dinámica		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El listado de los sensores no debe ser estática de forma que la inclusión de un nuevo sensor en la red de sensores no implique una reprogramación de la aplicación		

Tabla 7: RU-C02

ID: RU-C03			
Nombre:	Conexión servidor		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	La aplicación deberá mostrar un *popup* cuando se establezca la conexión con el *backend*		

Tabla 8: RU-C03

ID: RU-C04			
Nombre:	Detener servicio		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El usuario podrá detener en cualquier momento el		

	servicio de la aplicación de forma que no se establezca activa ninguna conexión

Tabla 9: RU-C04

ID: RU-C05			
Nombre:	Detener notificaciones		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El usuario podrá activar/desactivar las notificaciones de la aplicación. La deshabilitación de las notificaciones no implica la parada del servicio, por lo que no llegaran notificaciones al usuario pero la aplicación seguirá recibiendo actualizaciones desde el servidor		

Tabla 10: RU-C05

ID: RU-C06			
Nombre:	Gráficas		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El sistema debe permitir mostrar una gráfica para cada sensor donde se muestre si ha habido activaciones en un intervalo de tiempo dado.		

Tabla 11: RU-C06

ID: RU-C07			
Nombre:	Parámetro Gráficas		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	Las gráficas pueden ser parametrizadas indicando el intervalo de tiempo utilizado para las gráficas (1,5 o 15 minutos).		

Tabla 12: RU-C07

ID: RU-C08			
Nombre:	Servidor		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El nombre o dirección IP del servidor puede ser editable de forma que un cambio de servidor no implique reprogramación del aplicativo.		

Tabla 13: RU-C08

ID: RU-C09			
Nombre:	Topic		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El nombre del *topic* al que se suscribe la aplicación para obtener los datos de los sensores debe ser configurable.		

Tabla 14: RU-C09

ID: RU-C10			
Nombre:	Log		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	Todos los datos intercambiados entre el servidor y la aplicación serán almacenados		

Tabla 15: RU-C10

Los requisitos de restricción son:

ID: RU-R01			
Nombre:	Idioma		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	El idioma de la aplicación será el ingles		

Tabla 16: RU-R01

ID: RU-R02			
Nombre:	Android		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	La versión mínima de sistema de Android sobre la que podrá ser ejecutada la aplicación será la 2.1		

Tabla 17: RU-R02

ID: RU-R03			
Nombre:	Vertical / Horizontal		
Necesidad:	[ ]Esencial [X]Deseable [ ]Opcional		
Prioridad:	[ ]Alta [X]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario
Descripción:	La aplicación se podrá visualizar de forma vertical u horizontal		

Tabla 18: RU-R03

5.1.2 Requisitos software

En este otro grupo de requisitos se encuentran los que en base a los requisitos de usuario, definen como se comporta el sistema de forma más consistente y completa.

Los requisitos funcionales son:

ID: RS-F01			
Nombre:	Información sensores		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Cada sensor tendrá un identificador único, el estado del sensor y la fecha de la última activación		

Tabla 19: RS-F01

ID: RS-F02			
Nombre:	Lista sensores		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se irán mostrando en una lista de forma dinámica los sensores y su estado según se vayan recibiendo la información de estos.		

Tabla 20: RS-F02

ID: RS-F03			
Nombre:	Estado conexión		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	La aplicación mostrara cuando se realiza la conexión al servidor y sobre que servidor se realiza		

Tabla 21: RS-F03

ID: RS-F04			
Nombre:	Servicio		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	La aplicación debe ejecutarse como un servicio.		

Tabla 22: RS-F04

ID: RS-F05			
Nombre:	Detener servicio		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Debe ser posible detener el servicio de la aplicación		

Tabla 23: RS-F05

ID: RS-F06			
Nombre:	*Home \ Exit*		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Existirán dos botones para salir de la aplicación: Uno el *Home* dejara la aplicación en segundo plano y ejecutándose el servicio, el otro el *exit* cerrará la conexión con el servidor, detendrá el servicio y cerrará la aplicación.		

Tabla 24: RS-F06

ID: RS-F07			
Nombre:	Notificación		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se deberá poder configurar las notificaciones del sistema pudiéndose activar o desactivar éstas.		

Tabla 25: RS-F07

ID: RS-F08			
Nombre:	Gráfica		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Seleccionando un sensor se abrirá una nueva ventana con una gráfica sobre las activaciones del sensor		

Tabla 26: RS-F08

ID: RS-F09			
Nombre:	Preferencias		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se guarda en el sistema las preferencias del usuario respecto a la habilitación/deshabilitación de las notificaciones, parametrización de las gráficas y configuración del servidor y *topic*		

Tabla 27: RS-F09

ID: RS-F10			
Nombre:	Protocolo de comunicación		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Analista

Descripción:	Para el intercambio de información se usara el protocolo MQTT

Tabla 28: RS-F10

ID: RS-F11			
Nombre:	Fichero log		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se generara una fichero de logs con toda la información de la comunicación entre el servidor y la aplicación (estado de la conexión, eventos recibidos, etc…)		

Tabla 29: RS-F11

Los requisitos No funcionales son:

ID: RS-NF01			
Nombre:	XML Idioma		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema deberá de contener un archivo xml con las cadenas de texto en inglés.		

Tabla 30: RS-NF01

ID: RS-NF02			
Nombre:	Versión SDK Android		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	El sistema deberá de implementarse con el SDK 7 como versión mínima, para garantizar compatibilidad entre versiones.		

Tabla 31: RS-NF02

ID: RS-NF03			
Nombre:	BBDD Sensores		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se generara una BBDD Local (SQLite) para el listado de los sensores así como para almacenar los últimos datos recibidos de cada uno de los sensores		

Tabla 32: RS-NF03

ID: RS-NF04			
Nombre:	BBDD Gráficas		
Necesidad:	[X]Esencial [ ]Deseable [ ]Opcional		
Prioridad:	[X]Alta [ ]Media [ ]Baja	**Estabilidad**:	Alta
Verificabilidad:	[X]Alta [ ]Media [ ]Baja	**Fuente:**	Usuario/Analista
Descripción:	Se generara una BBDD Local (SQLite) para guardar los datos necesarios a la hora de generar las gráficas.		

Tabla 33: RS-NF04

5.1.3 Trazabilidad RS -> RU

	RUC01	RUC02	RUC03	RUC04	RUC05	RUC06	RUC07	RUC08	RUC09	RUC10	RUR01	RUR02	RUR03
RSF01	X												
RSF02		X											
RSF03			X										
RSF04				X									
RSF05				X									
RSF06													
RSF07					X								
RSF08						X							
RSF09							X	X	X				
RSF10													
RSF11										X			
RSNF01											X		
RSNF02												X	
RSNF03		X											
RSNF04						X							

Tabla 34: Trazabilidad RS -> RU

5.1.4 Casos de uso

Un caso de uso es una secuencia de interacciones que se desarrollan entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

En este apartado se van a describir las secuencias de uso más comunes que se ejecutarán en el sistema. Se dividirá este apartado en dos, en una de ellas se detallará de forma textual el caso de uso, y en otra de ellas se explicará de forma gráfica.

5.1.4.1 Descripción Textual

En la descripción textual de cada uno de los casos de uso, se especificarán los campos que se muestran a continuación:

- Identificador (ID): Determinará de forma unívoca cada uno de los casos de uso que se detallarán a continuación. El formato a seguir será: CU-XX siendo las X números comprendidos entre 0 y 9. Se comenzará por el 01 y se irá incrementando una unidad por cada caso de uso nuevo.

- Título: Nombre descriptivo acerca del caso de uso.

- Actor: Siempre hará referencia a un único tipo de usuario, ya que en el sistema no existen distintos perfiles de usuario.

- Descripción: En este campo se realizará una breve descripción del caso de uso a comentar.

- Precondiciones: Condiciones previas que se deberán cumplir para poder ejecutar el caso de uso.

- Postcondiciones: Condiciones que se producirán tras la ejecución del caso de uso.

- Escenario principal: Trata de la secuencia común de interacciones ordenadas, especificando la interacción del usuario con el sistema.

- Escenario alternativo: Describirá la ejecución del caso de uso con condiciones de error o caminos de decisión distintos al principal.

Los casos de uso textuales son los siguientes:

ID: CU-01			
Titulo:	Listado sensores	**Actor:**	Usuario
Descripción:	Se muestra en la interfaz gráfica, el listado de sensores así como estado e información		
Precondiciones:	Que la aplicación ya haya establecido en algún momento conexión con el servidor y recibidos datos de algún sensor.		
Postcondiciones:	Ninguna		
Escenario principal:	1. Usuario ejecuta aplicación 2. Se muestra la lista de los sensores		
Escenario alternativo:	1a. El usuarios se encuentra en la aplicación en otra pantalla (gráficas o preferencias) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo		

Tabla 35: CU - 01

ID: CU-02			
Titulo:	Establecer conexión	**Actor:**	Usuario
Descripción:	El usuario ejecuta la aplicación y se establece la conexión con el servidor		
Precondiciones:	Que exista el fichero de preferencias con un servidor y un *topic* asignados y que el servidor se encuentre disponible		
Postcondiciones:	Ninguna		
Escenario principal:	1. Usuario ejecuta aplicación 2. Se muestra el listado de sensores 3. Se muestra un *popup* con los datos de la conexión realizada		

Escenario alternativo:	1a. El usuario se encuentra en la aplicación en otra pantalla (gráficas o preferencias) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo 3a. No se muestra el *popup* de la conexión debido a que la conexión no se ha podido establecer

Tabla 36: CU-02

ID: CU-03			
Titulo:	Gráficas	**Actor:**	Usuario
Descripción:	El usuario ejecuta la aplicación y dentro de la lista de sensores elije uno para mostrar su gráfica		
Precondiciones:	Que exista al menos un sensor en la lista de sensores		
Postcondiciones:	Ninguna		
Escenario principal:	1. Usuario ejecuta aplicación 2. Se muestra el listado de sensores 3. El usuario selecciona un sensor 4. Se muestra en una nueva ventana la gráfica de dicho sensor		
Escenario alternativo:	1a. El usuario se encuentra en la aplicación en otra pantalla (gráficas o preferencias) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo		

Tabla 37: CU-03

ID: CU-04			
Titulo:	Preferencias	**Actor:**	Usuario
Descripción:	El usuario ejecuta la aplicación y edita las preferencias		
Precondiciones:	Ninguna		
Postcondiciones:	En caso que se modifiquen las preferencias de servidor o *topic* el usuario debe cerrar y volver abrir la aplicación para que los cambios surjan efectos		
Escenario	1. Usuario ejecuta aplicación		

principal:	2. El usuario pulsa el botón *Menú* del dispositivo móvil 3. Dentro de las opciones desplegadas el usuario elije *Preferences* 4. Se despliega el menú con las preferencias 5. El usuario edita las preferencias a su gusto
Escenario alternativo:	1a. El usuarios se encuentra en la aplicación en otra pantalla (gráficas) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo

Tabla 38: CU-04

ID: CU-05			
Titulo:	Minimizar aplicación	**Actor:**	Usuario
Descripción:	El usuario ejecuta la aplicación y minimiza la aplicación, dejando la aplicación en segundo plano		
Precondiciones:	Ninguna		
Postcondiciones:	El servicio sigue corriendo y la conexión con el servidor sigue establecida. El usuario si tiene habilitadas las notificaciones puede seguir recibiendo notificaciones del estado de los sensores		
Escenario principal:	1. Usuario ejecuta aplicación 2. El usuario pulsa el botón *Menú* del dispositivo móvil 3. Dentro de las opciones desplegadas el usuario elije *Home* 4. Se muestra el menú principal del dispositivo.		
Escenario alternativo:	1a. El usuarios se encuentra en la aplicación en otra pantalla (gráficas o preferencias) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo		

Tabla 39: CU-05

ID: CU-06			
Titulo:	Salir aplicación	**Actor:**	Usuario
Descripción:	El usuario ejecuta la aplicación y sale de la aplicación.		
Precondiciones:	Ninguna		
Postcondiciones:	Se realiza la desconexión con el servidor, se liberan los recursos y se cierra la aplicación		
Escenario principal:	1. Usuario ejecuta aplicación 2. El usuario pulsa el botón *Menú* del dispositivo móvil 3. Dentro de las opciones desplegadas el usuario elije *Exit* 4. Se muestra el menú principal del dispositivo.		
Escenario alternativo:	1a. El usuarios se encuentra en la aplicación en otra pantalla (gráficas o preferencias) en tal caso deberá dar a la tecla *back* 1b. El usuario recibe una notificación del sistema, y selecciona la notificación 2a. No se muestra ningún dato debido a que la aplicación no ha recibido ningún dato desde que se instalo		

Tabla 40: CU-06

5.1.4.2 Descripción grafica

Tras describir los casos de uso de la aplicación se procede a mostrar los distintos diagramas de casos de uso. Los casos de uso se representan por elipses y se muestran como parte del contexto o aplicación que está siendo modelado, los actores sin embargo se externalizan.

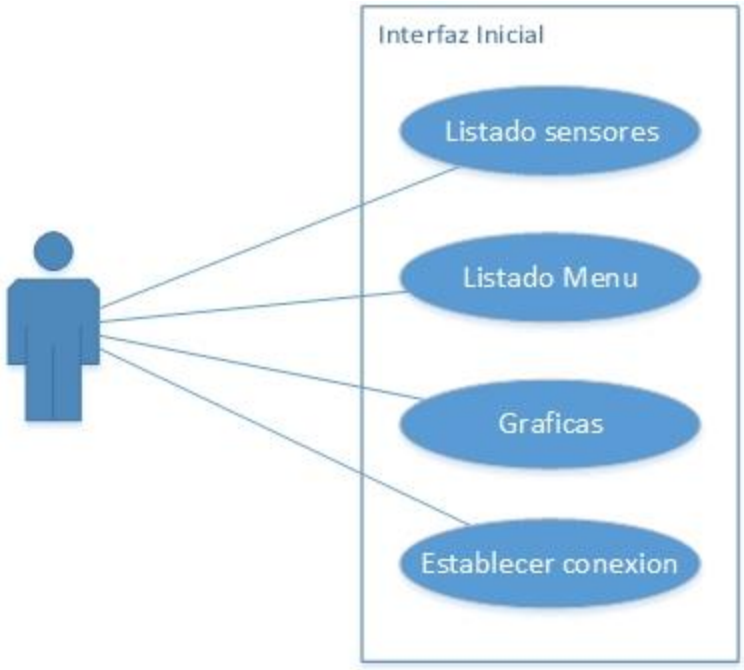


Ilustración 20: Caso de Uso – Interfaz inicial

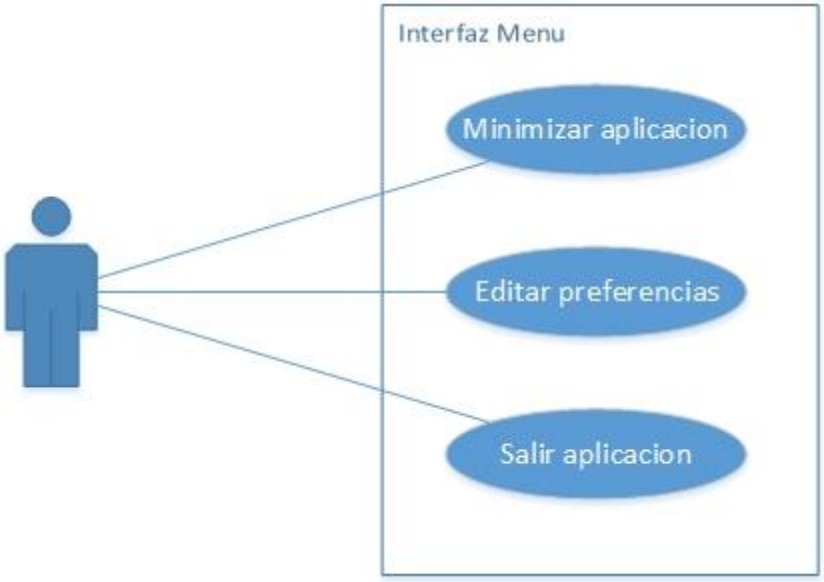


Ilustración 21: Caso de Uso – Interfaz Menú

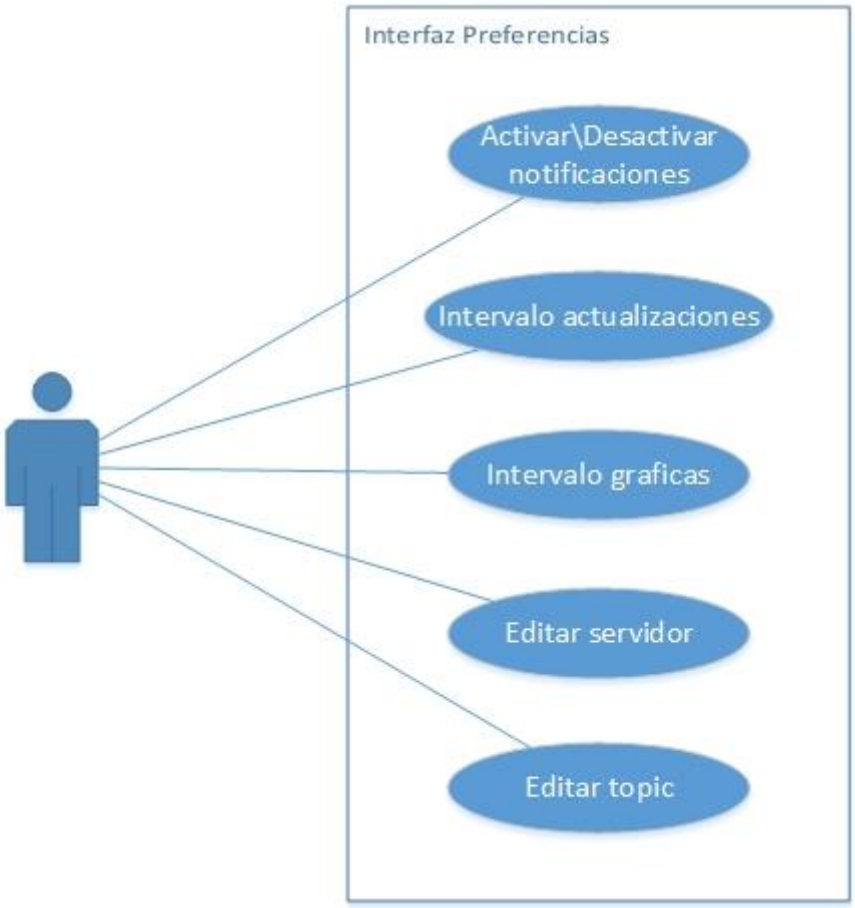


Ilustración 22: Caso de Uso – Interfaz preferencias

5.2 Diseño

El objetivo de esta fase es la de modelar el sistema basándose en la salida de la fase anterior (análisis), para posteriormente implementar el proyecto.

A continuación se muestra un diseño a alto nivel de la totalidad de la plataforma:

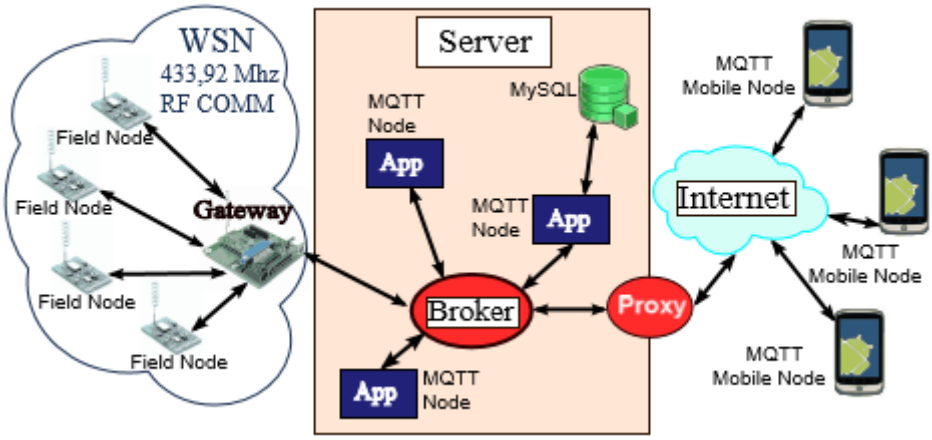


Ilustración 23: Diseño alto nivel plataforma

5.2.1 Arquitectura

Una vez estudiados los patrones más conocidos y utilizados en la ingeniería del software, se comprueba que el patrón más acorde a este sistema sería un patrón "Modelo-Vista-Controlador (MVC)". Este patrón está formado por tres partes diferenciadas:

- Modelo: Es el encargado de tratar toda la lógica del negocio, representando específicamente toda la información con la que el sistema va a trabajar y notificando a la vista los cambios que se producen en el sistema. Es el "motor" de la aplicación, en este proyecto el modelo estará compuesto por las BBDD SQLite de los sensores y las gráficas.

- Vista: Es la parte del sistema con la que interacciona el usuario, recibiendo y mostrando datos en pantalla. Normalmente se cuenta con más de una interfaz, por lo que también se encarga de navegar entre las distintas interfaces. Es la "cara" de la aplicación, en este proyecto la vista estará formada por los distintos archivos XML necesarios en Android para generar la interfaz de usuario.

- Controlador: Procesa las peticiones que realiza el usuario y las envía a la lógica del sistema para que las trate y devuelva un resultado la información adecuada para mostrársela al usuario. Es el "transportador"

de la aplicación, en este proyecto serán las clases de java programadas para controlar la actividad y el servicio de la aplicación.

A continuación se muestra el patrón MVC orientado a aplicaciones Android:

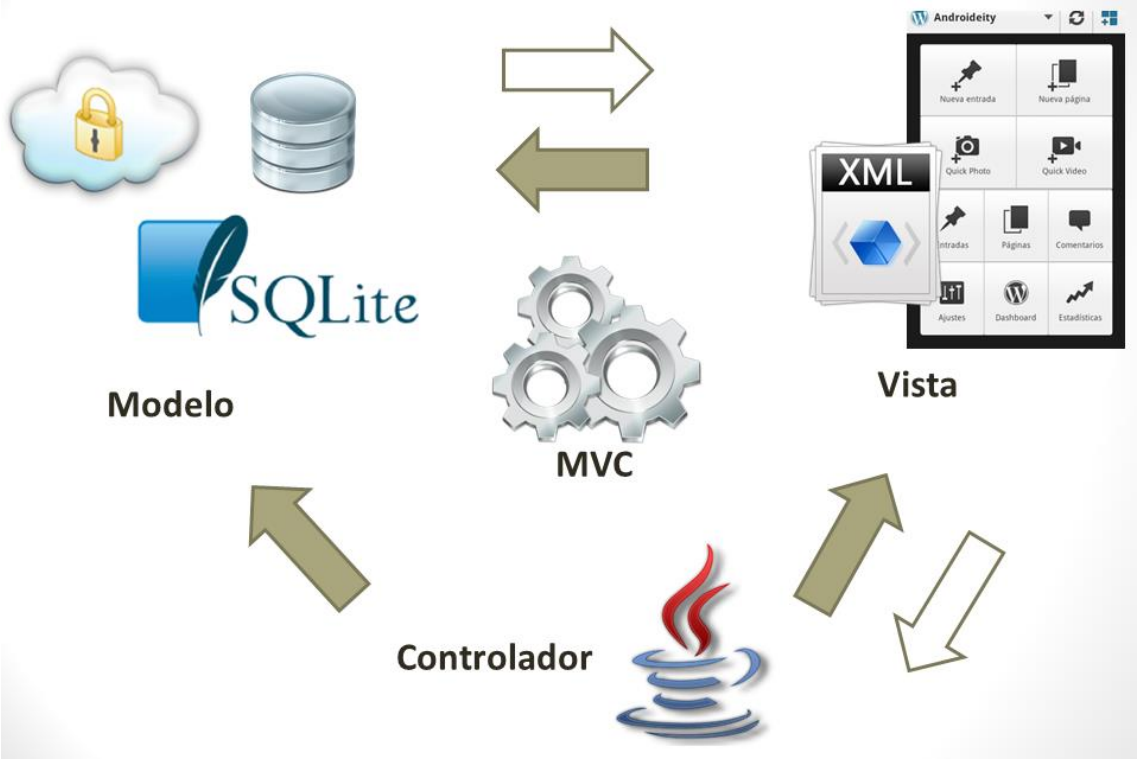


Ilustración 24: Patrón MVC

Gracias a este tipo de modelos se puede obtener una aplicación escalable, construida mediante perfiles especializados que se encargarán de desarrollar cada uno de los componentes. De esta forma el diseñador gráfico no necesitará más que saber XML desentendiéndose de la lógica de negocio, y lo mismo ocurrirá con el encargado del modelado de datos y el del controlador.

Un flujo clásico de esta arquitectura podría ser el siguiente:

1. El usuario comienza interactuando con la aplicación. El encargado es la vista (mostrando la lista de sensores)
2. El controlador recibe una acción (El usuario quiere ver la gráfica de un sensor).
3. El modelo es llamado para acceder a las últimas activaciones de dicho sensor.

4. Una vez obtenidas todas las activaciones, el controlador tomará partida para llamar a la vista y generar la gráfica.
5. El usuario obtendría la nueva interfaz con la gráfica de las últimas activaciones del sensor que eligió.

5.2.2 Modelado del sistema

El modelado de un sistema sirve como vista gráfica de la información que se especificó con anterioridad en la fase de requisitos. Se podría decir que es una abstracción o simplificación del sistema real a desarrollar, que sirve como es obvio para poder comprenderlo mejor. Dicho modelo estará formado por distintos diagramas o vistas parciales del modelo que servirán como base para la futura implementación.

A lo largo de esta sección se mostrarán los distintos tipos de diagramas empleados para modelar el sistema entre los que se encuentran: diagrama de flujo, diagrama de secuencia y diagrama de clases. Por último se mostrará la trazabilidad entre los requisitos software y los componentes de la arquitectura.

5.2.2.1 Diagramas de flujo

Los diagramas de flujo son una manera de representar visualmente el flujo de datos a través de sistemas de tratamiento de información.

Un diagrama de flujo u organigrama es una representación diagramática que ilustra la secuencia de las operaciones que se realizarán para conseguir la solución de un problema. Los diagramas de flujo se dibujan generalmente antes de comenzar la implantación. Los diagramas de flujo facilitan la comunicación entre los programadores y la gente del negocio.

Se mostrará un diagrama de flujo por cada una de las posibles acciones básicas o procesos que se puede desempeñar dentro del sistema.

A continuación se muestra la simbología usada en los diagramas:

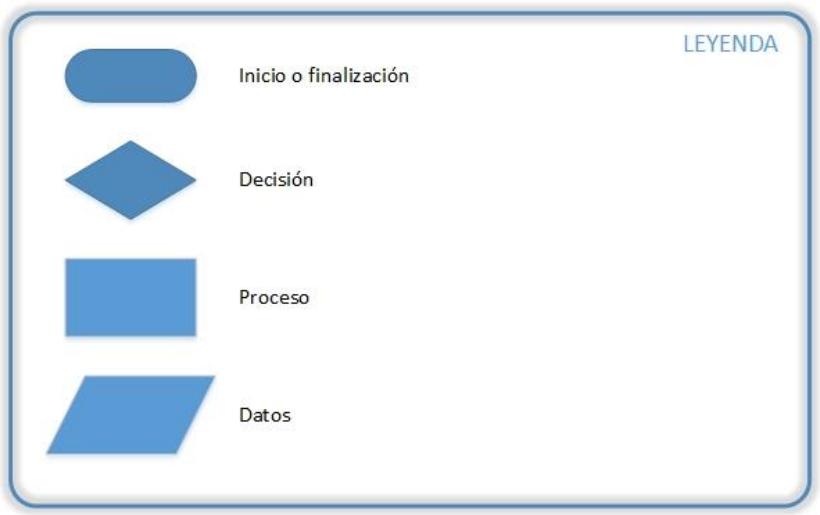


Ilustración 25: Diagrama de flujo – Leyenda

Diagrama de flujo correspondiente a mostrar la gráfica de un sensor:

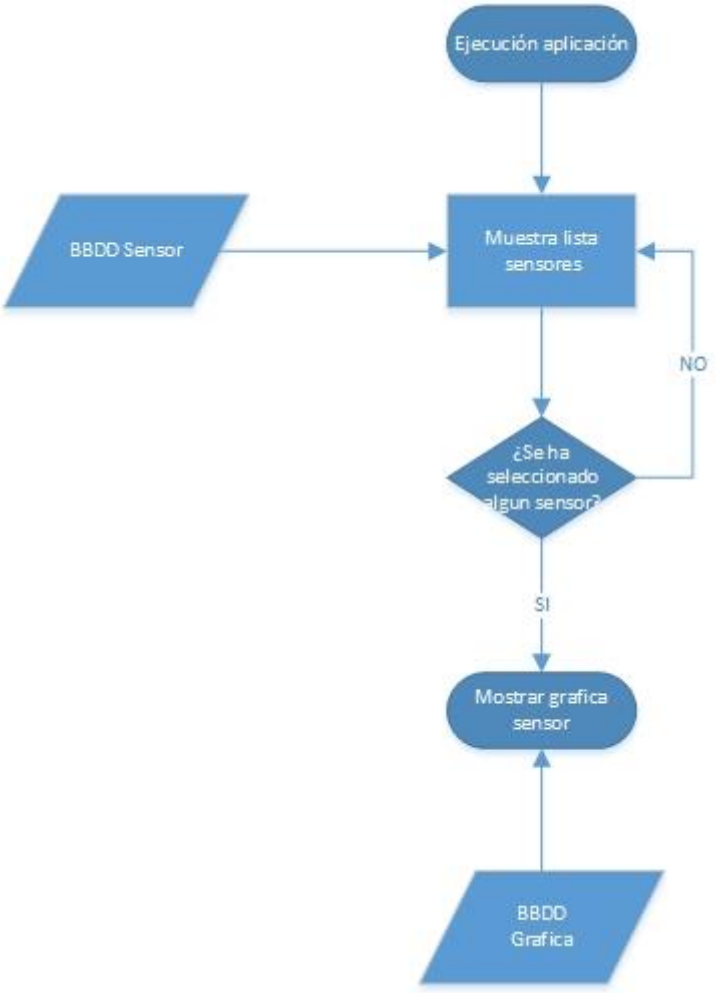


Ilustración 26: Diagrama de flujo – Mostrar grafica de un sensor

Diagrama de flujo correspondiente a editar las preferencias:

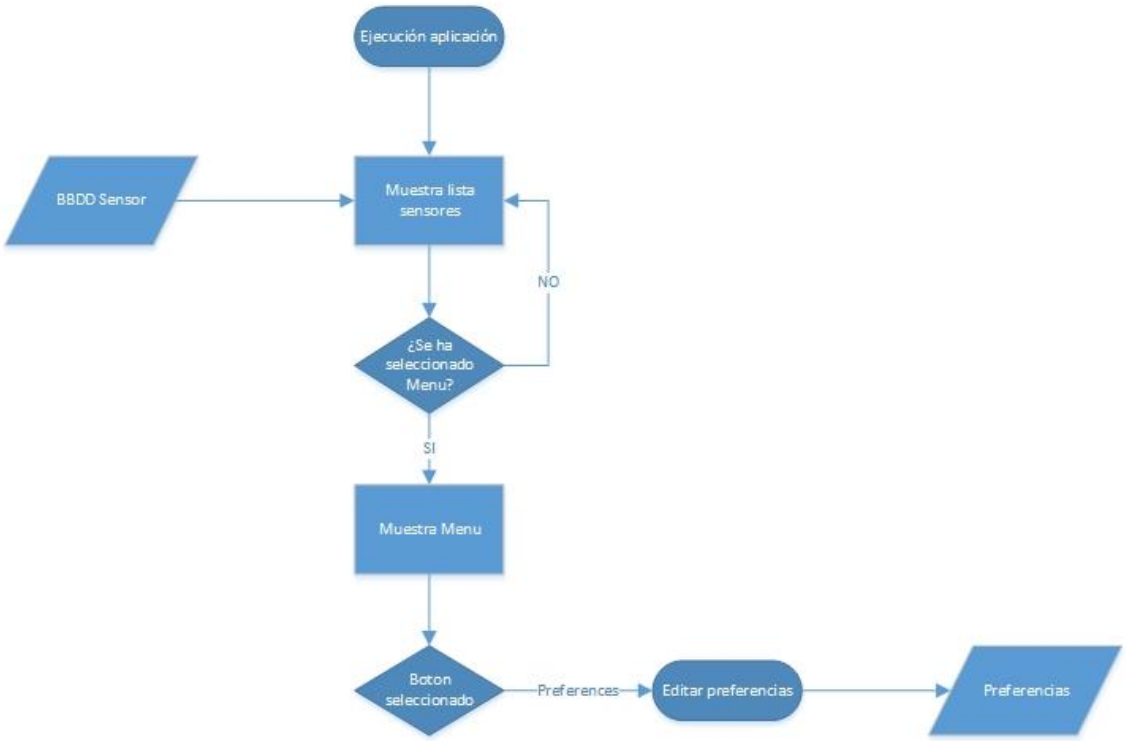


Ilustración 27: Diagrama de flujo – Editar preferencias

Diagrama de flujo correspondiente a salir o minimizar la aplicación:

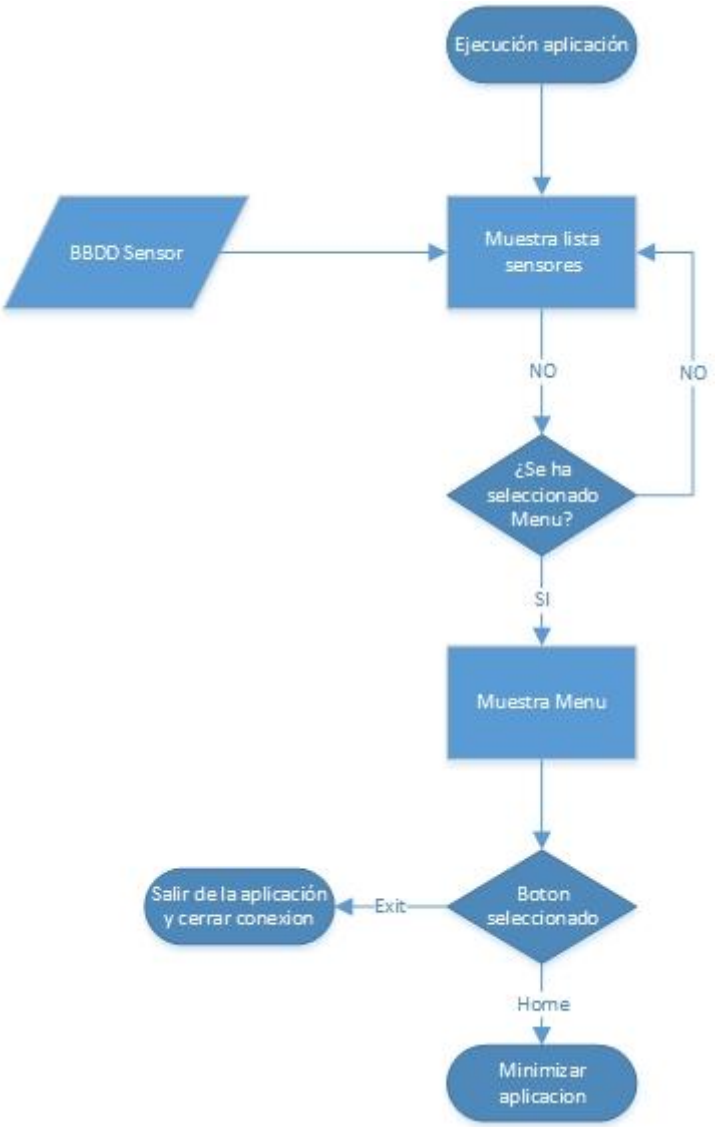


Ilustración 28: Diagrama flujo – Salir o minimizar aplicación

Diagrama de flujo correspondiente a la llegada de una publicación:

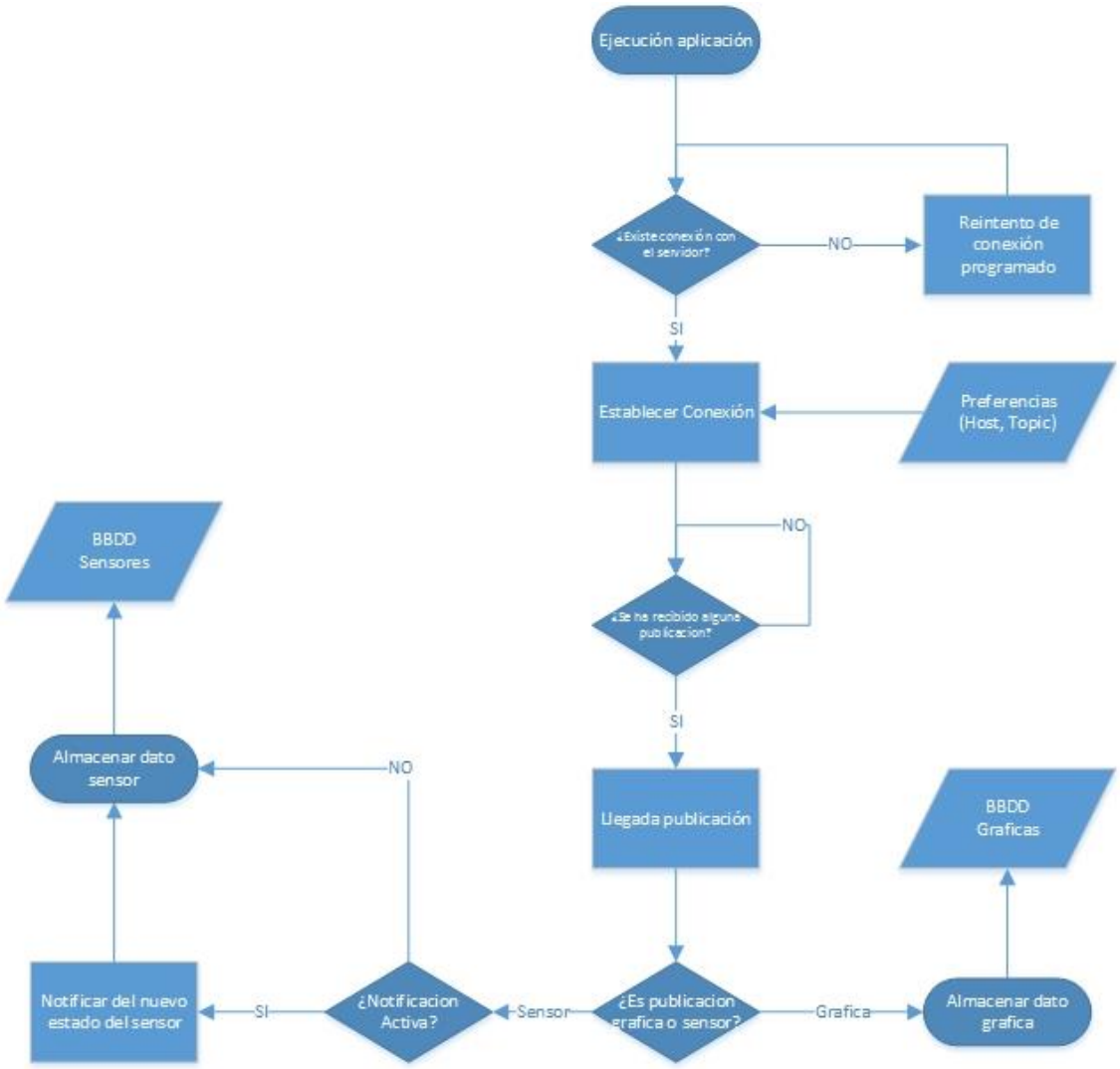


Ilustración 29: Diagrama flujo – Llegada publicación

5.2.2.2 Diagrama de clases

El diagrama de clases del proyecto se puede apreciar en la siguiente ilustración:

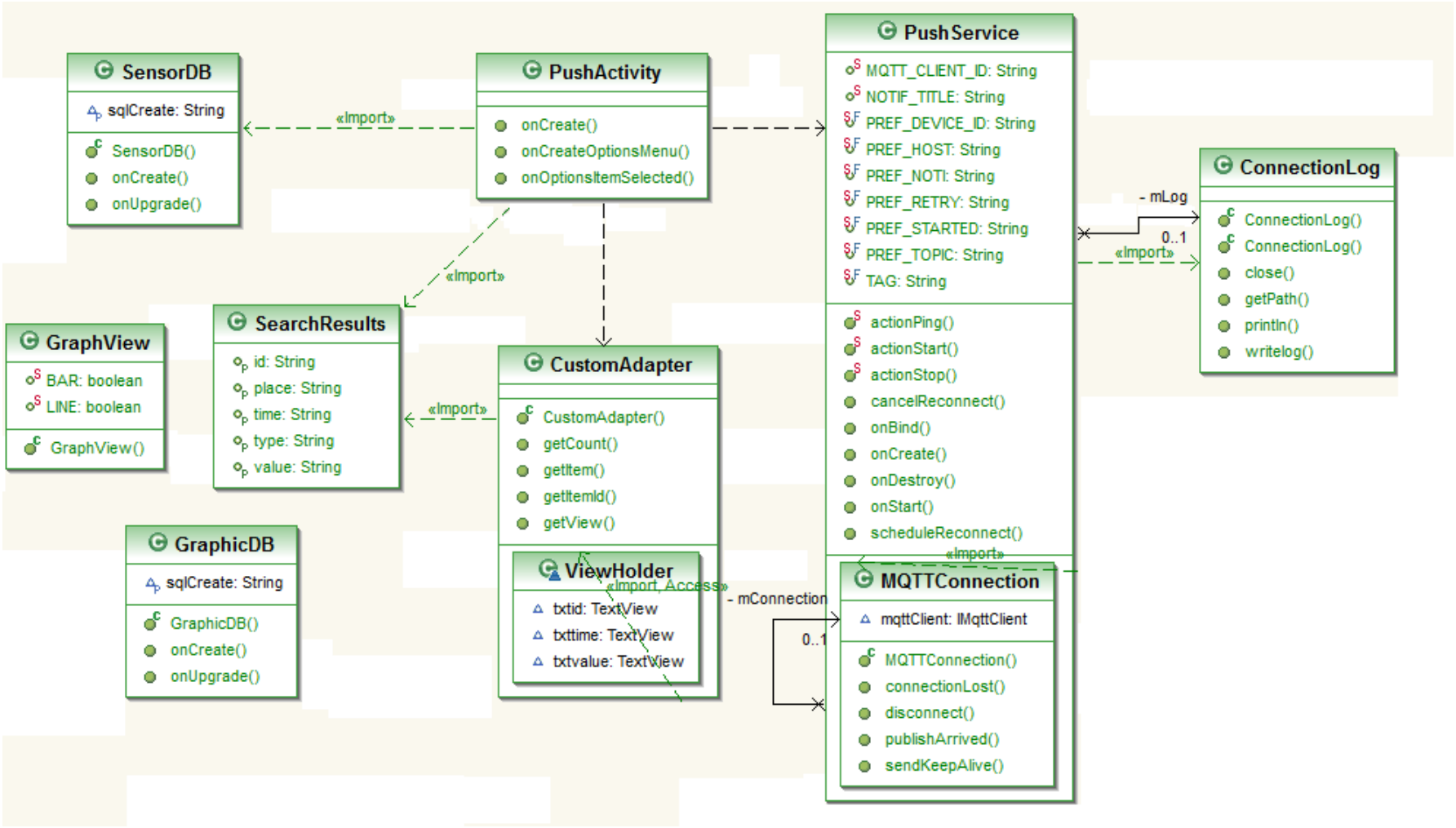


Ilustración 30: Diagrama de clases

5.2.2.3 Diagramas de secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. Mientras que el diagrama de casos de uso permite el modelado de una vista *business* del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario y mensajes intercambiados entre los objetos.

Los casos de usos que se van a modelar son:

- CU-01: Listado sensores
- CU-02: Establecer conexión
- CU-03: Gráficas
- CU-04: Preferencias
- CU-05: Minimizar
- CU-06: Salir

Diagrama de secuencia – Listado sensores:

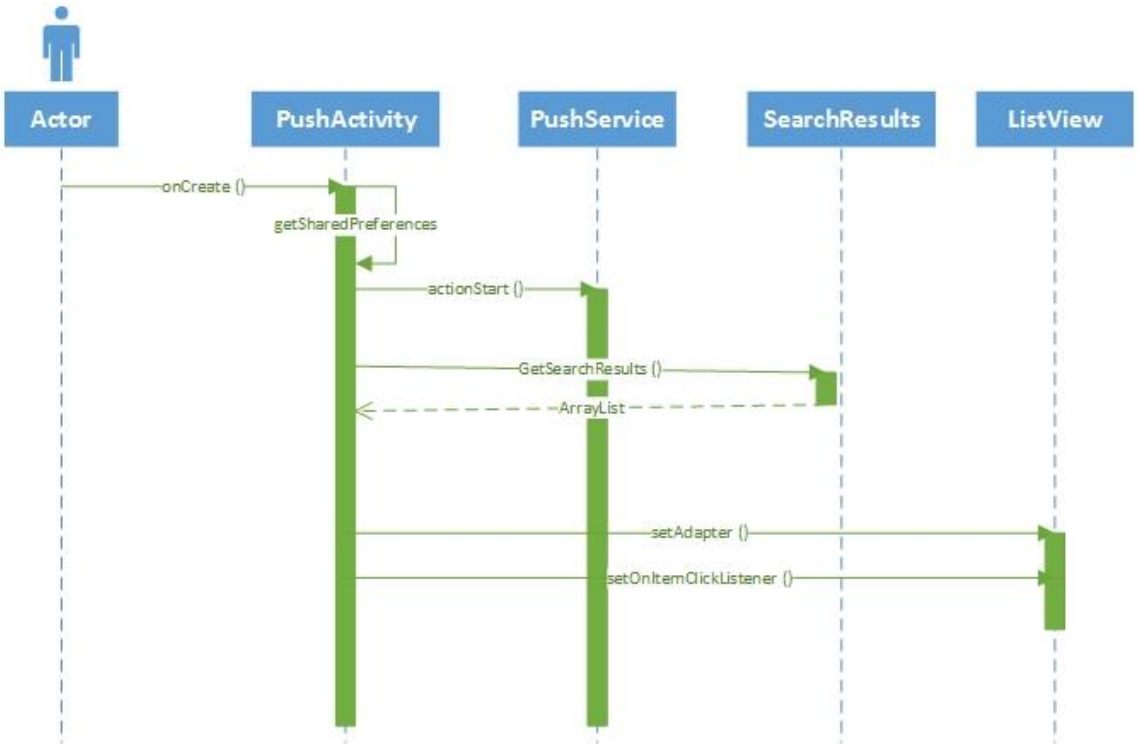


Ilustración 31: Diagrama de secuencia – Listado de sensores

Diagrama de secuencia – Establecer conexión:

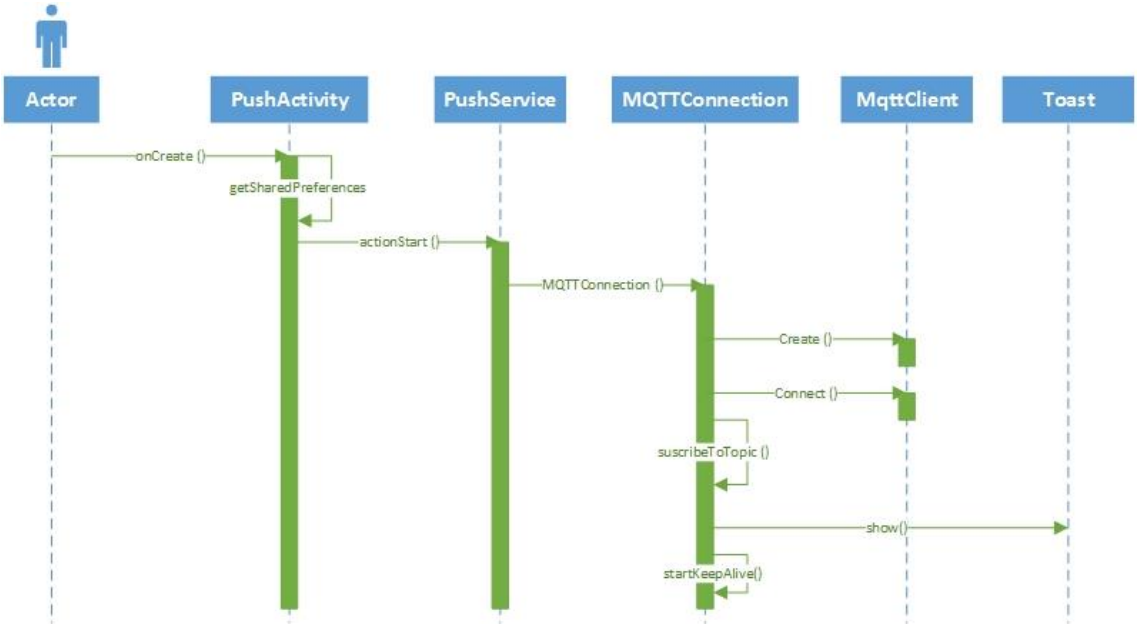


Ilustración 32: Diagrama secuencia – Establecer conexión

Diagrama de secuencia – Gráficas:

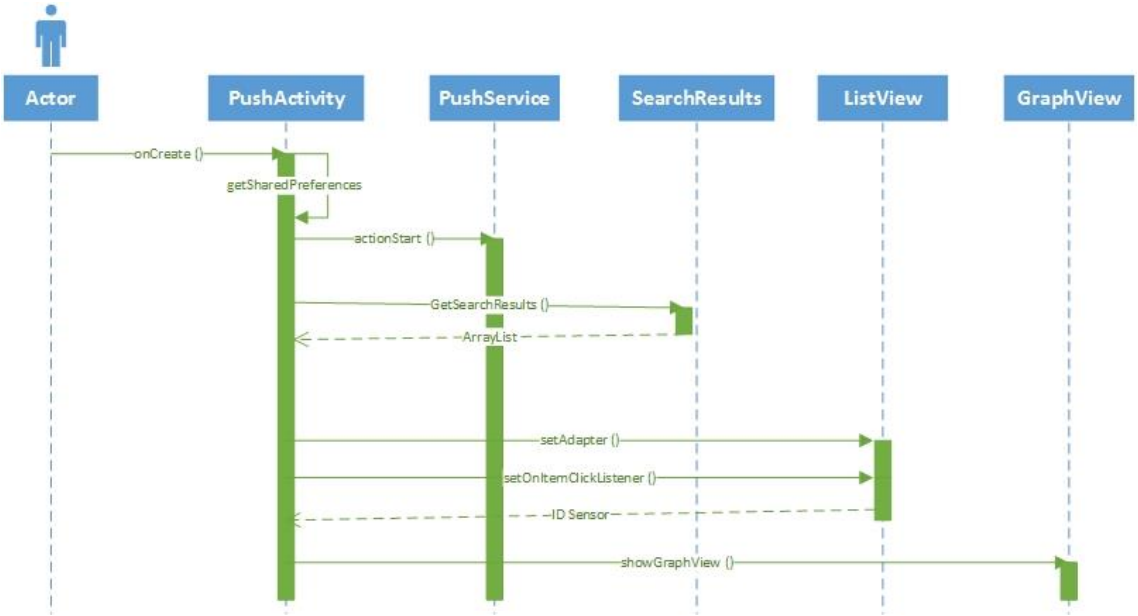


Ilustración 33: Diagrama secuencia – Gráficas

Diagrama de secuencia – Preferencias:

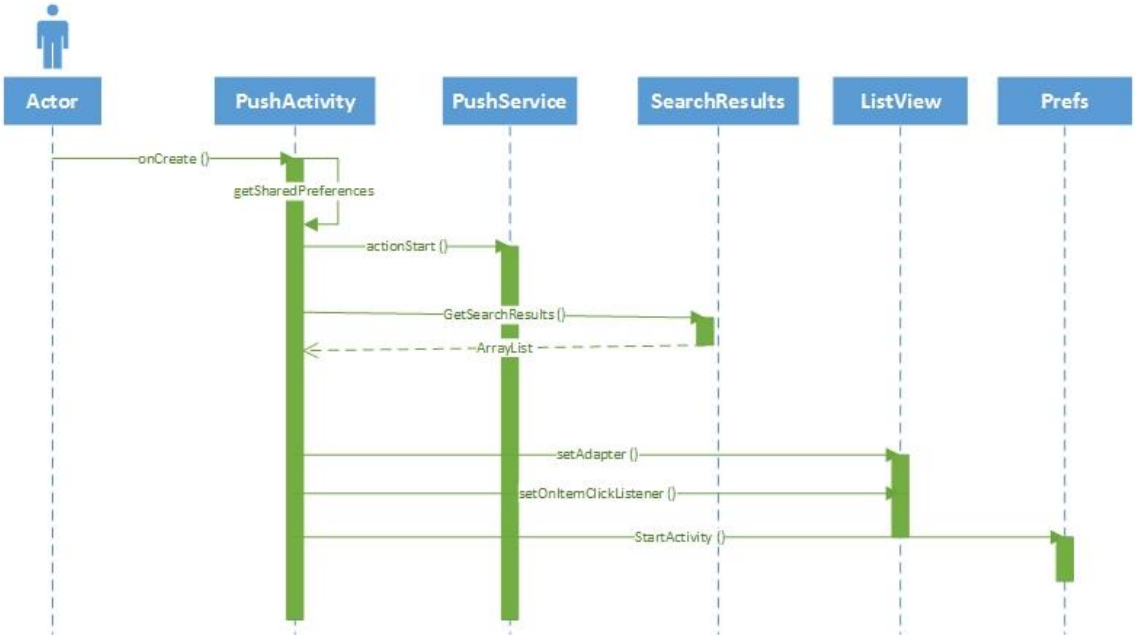


Ilustración 34: Diagrama secuencia – Preferencias

Diagrama de secuencia – Minimizar:

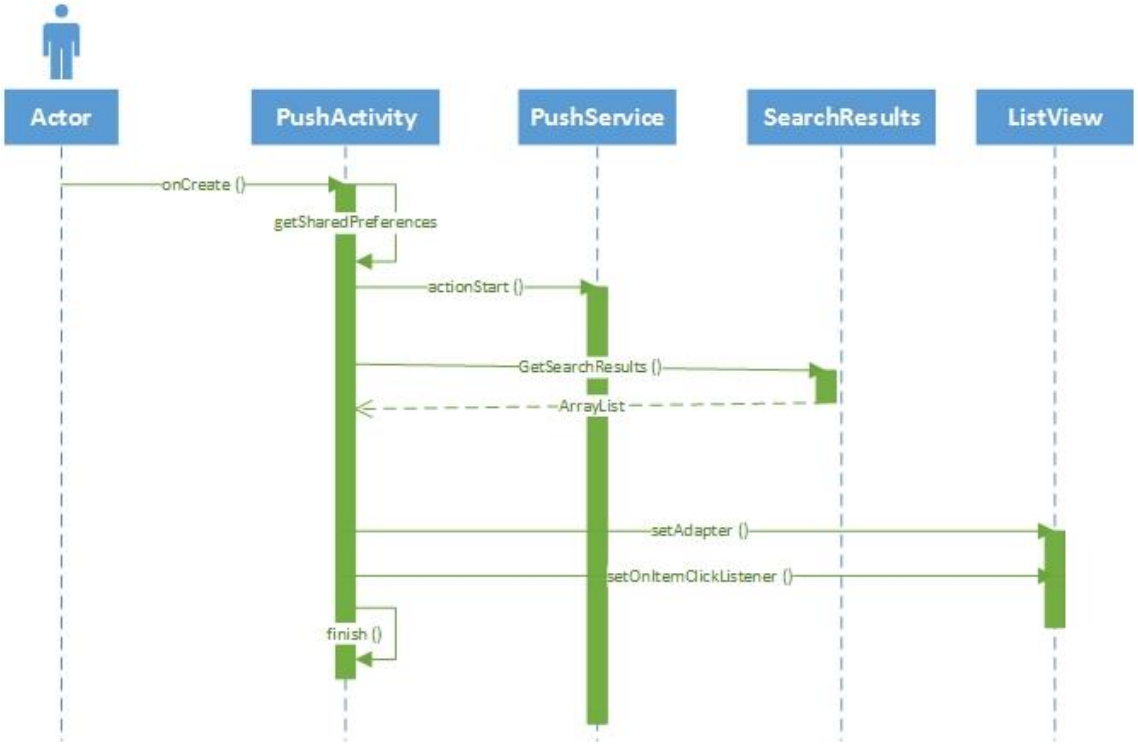


Ilustración 35: Diagrama secuencia – Minimizar

Diagrama de secuencia – Salir:

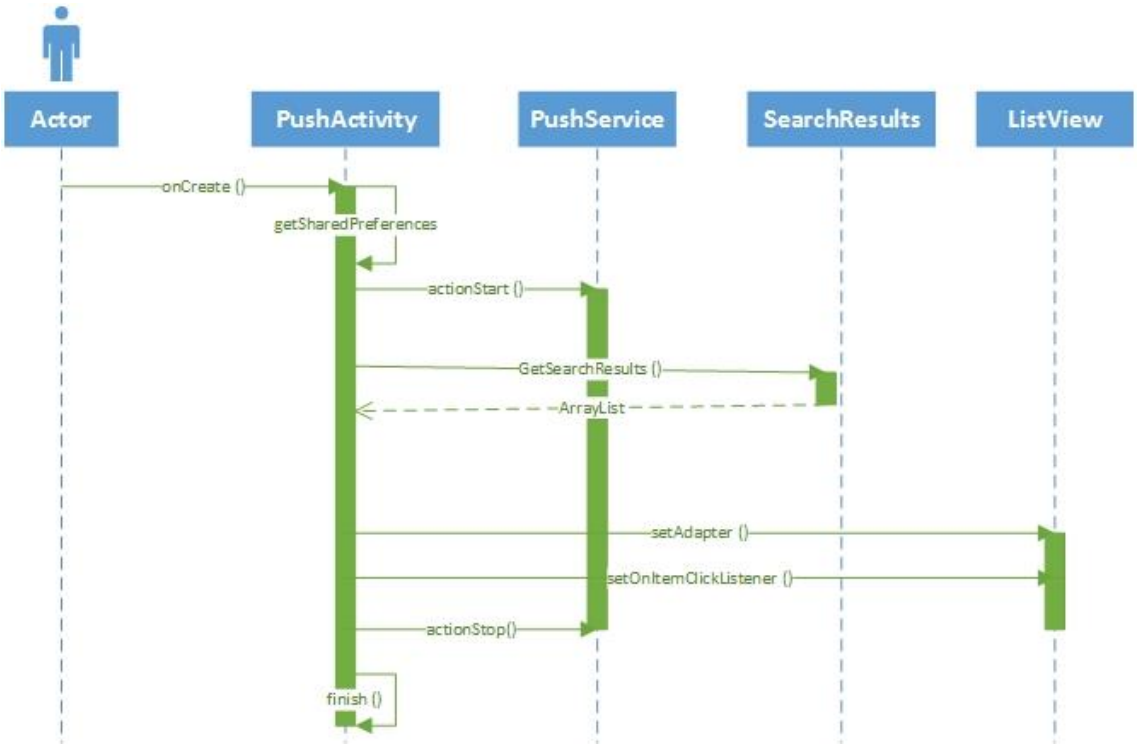


Ilustración 36: Diagrama secuencia – Salir

5.2.2.4 Trazabilidad Requisitos – Arquitectura

Esta arquitectura debe cumplir con la premisa de que todo requisito de software debe encontrarse en, al menos, un componente de la arquitectura, por lo que es necesario desarrollar un mecanismo de comprobación. Dicha comprobación se puede observar en la matriz de trazabilidad que se muestra a continuación:

	Modelo	Vista	Controlador
RUC01	X	X	
RUC02	X	X	
RUC03			X
RUC04		X	X
RUC05		X	X
RUC06	X	X	
RUC07			X

RUC08		X	X
RUC09		X	X
RUC10		X	X
RUR01		X	
RUR02			X
RUR03		X	
RSF01	X		
RSF02	X	X	
RSF03			X
RSF04			X
RSF05			X
RSF06		X	
RSF07			X
RSF08		X	
RSF09	X		
RSF10			X
RSF11			X
RSNF01		X	
RSNF02			X
RSNF03	X		
RSNF04	X		

Tabla 41: Trazabilidad Requisitos – Arquitectura

5.3 Implantación

En este apartado se tratará todo el proceso de codificación real del sistema, describiendo el lenguaje de programación usado y el entorno de programación en el que se ha desarrollado la aplicación además de explicar de forma detallada otras decisiones importantes tomadas a la hora de la implementación en código.

5.3.1 Lenguaje, entorno y estructura del proyecto

Para la implantación de la aplicación se ha utilizado el SDK de Android el cual facilita las librerías (API) y las herramientas necesarias para construir, testear y depurar aplicaciones para Android. El IDE elegido para el desarrollo de la aplicación ha sido Eclipse unido al *plugin Android Developer Tools* (ADT). Respecto al lenguaje de programación usado para la programación de la aplicación se ha usado Java para las clases y el lenguaje de XML para la creación de la interfaz de usuario.

A continuación se muestra la estructura del proyecto dentro del IDE Eclipse:

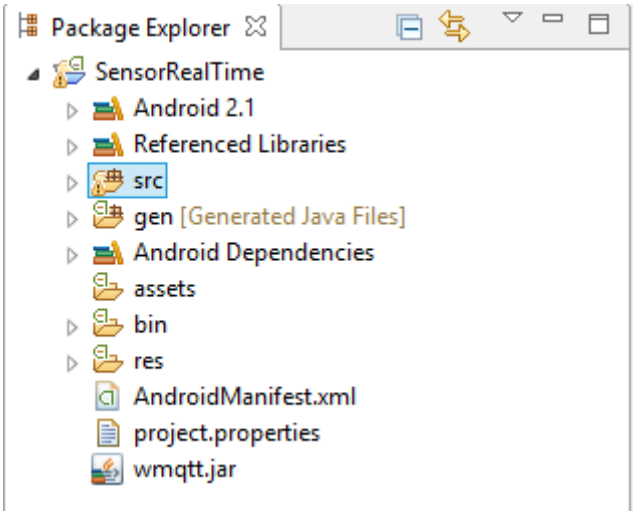


Ilustración 37: Estructura del proyecto

Se ha optado por generar un solo paquete com.SRT en el cual están creadas todas las clases necesarias para el desarrollo de la aplicación

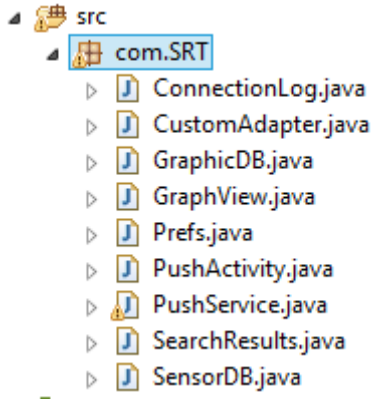


Ilustración 38: Paquete com.SRT

A continuación se describen la funcionalidad de cada una de las clases:

- ConnectionLog: Clase encargada de generar y escribir los eventos en el log generado por la aplicación.

Ilustración 39: Clase ConnectioLog

- CustomAdapter: Extiende a la clase auxiliar BaseAdapter, esta clase se encarga de mostrar la lista de sensores de forma dinámica

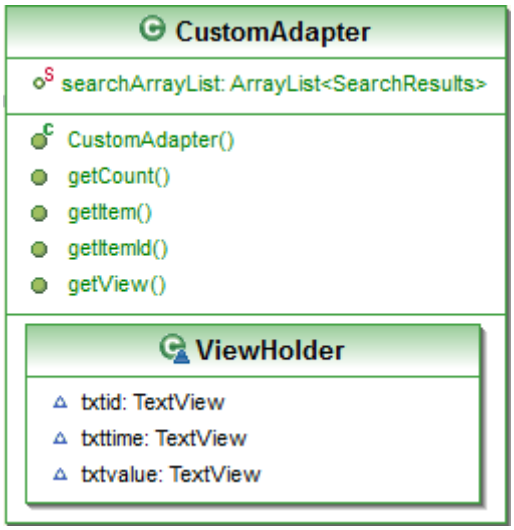


Ilustración 40: Clase CustomAdapter

- GraphicBD: Extiende a la clase auxiliar SQLiteOpenHelper y es la clase encargada de la creación de la BBDD de Gráficas

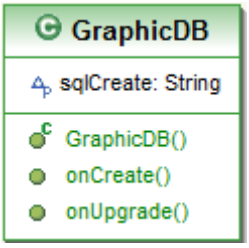


Ilustración 41: Clase GraphicDB

- GraphView: Extiende a la clase auxiliar View y se encarga de dibujar la gráfica de los sensores

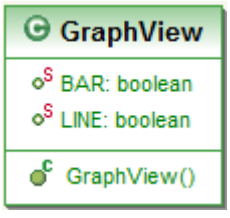


Ilustración 42: Clase GraphView

- Prefs: Extiende a la clase auxiliar PreferenceActivity y se encarga de crear el fichero de preferencias

Ilustración 43: Clase Prefs

- PushActivity: Extiende la clase auxiliar Activity, es la clase llamada en la ejecución de la aplicación, arranca el servicio, y gestiona las distintas interfaces de usuarios

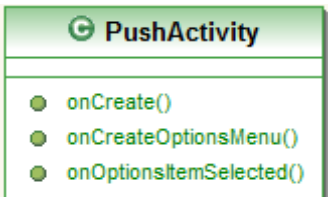


Ilustración 44: Clase PushActivity

- PushService: Extiende la clase auxiliar Service, es la clase que se encarga de gestionar el servicio y en la cual se realiza la conexión con el servidor, manteniendo activa la comunicación para poder recibir las publicaciones realizadas.

Ilustración 45: Clase PushService

- SearchResults: Clase que contiene los atributos que van a contener los sensores mostrados en la lista de sensores

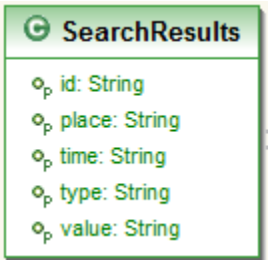


Ilustración 46: Clase SearchResults

- SensorDB: Extiende a la clase auxiliar SQLiteOpenHelper y es la clase encargada de la creación de la BBDD de los Sensores

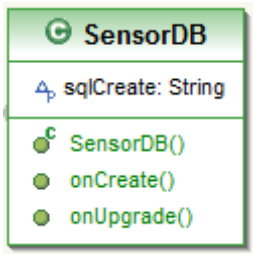


Ilustración 47: Clase SensorDB

En la carpeta *drawable* se encuentran las imágenes utilizadas para la aplicación (icono, *exit*, *home* y preferencias)

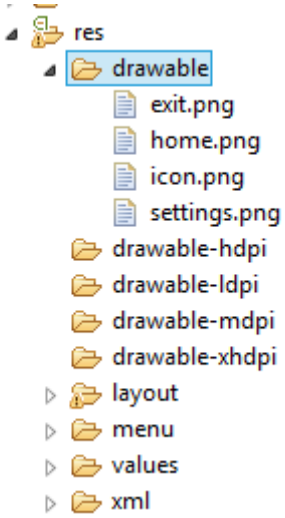


Ilustración 48: Imágenes

Además de las imágenes en la carpeta *resources* (res) se encuentran los XML utilizados para la generación de la interfaz de usuario así como para los literales mostrados en las distintas ventanas

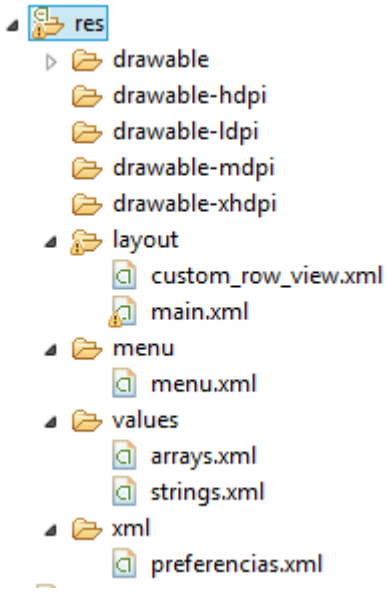


Ilustración 49: XMLs interfaz de usuario

Las librerías utilizadas en el proyecto han sido: la librería proporcionada por Android en su versión para sistemas operativos 2.1 y la librería wmqtt que implementa los métodos necesarios para hacer la comunicación con el protocolo MQTT.

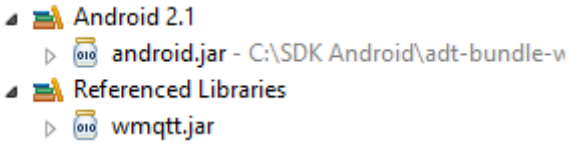


Ilustración 50: Librerías

5.3.2 DDBB

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre.

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será a través de una clase auxiliar llamada SQLiteOpenHelper, la utilizada en el proyecto.

Este motor de BBDD ha sido usado para las dos BBDD utilizadas por la aplicación que están almacenadas en el propio dispositivo, ya que como se ha comentado anteriormente no necesitan de servidor. A continuación se muestra los campos de los que se compone cada una de las BBDD

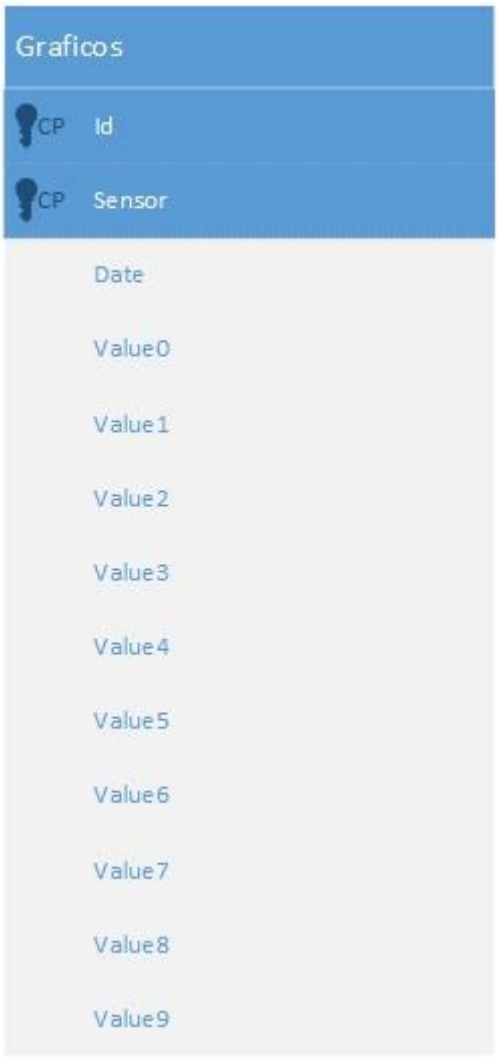


Ilustración 51: Tabla Gráficos

Ilustración 52: Tabla sensores

5.3.3 MQTT – Protocolo de comunicación

Para la comunicación con la parte servidor se ha decidido usar MQTT, el cual es un protocolo de mensajería abierto, simple, ligero y fácil de implementar.

Se ha decidido usar MQTT en vez de C2DM debido tanto a las características de MQTT como a las limitaciones de C2DM.

Características de MQTT:

- Diseñado para proporcionar bajas latencias
- Diseñado para enviar gran cantidad de mensajes de pequeño tamaño
- Optimizado para enviar el menor número de bytes
- Optimizado para usar un bajo consumo de energía
- Posibilidad de comunicación bidireccional
- Tres calidades de servicio para la entrega de mensajes

Limitaciones de C2DM:

- Necesario mínimo Android 2.2
- Requiere de la configuración de una cuenta de Google en el dispositivo
- Google limita el número de mensaje enviados por un remitente a un especifico dispositivo
- Sistema de comunicación unidireccional

- C2DM no ofrece ninguna garantía sobre la entrega o el orden de los mensajes.

La aplicación se suscribe a un *topic* (el cual puede ser modificado en las preferencias) y espera a que en la parte servidora se empiecen a publicar los datos referentes a ese *topic*, estos datos que se van a publicar son los datos de los sensores así como los últimos valores de los sensores para poder generar las gráficas.

La aplicación espera que los datos publicados por el servidor para cada uno de los sensores tengan el siguiente formato:

'3d2;Door;Outside;0;26/4/13 07:43:18:889'

Para los datos publicados para las gráficas se espera que tenga el siguiente formato:

T1;S3d2;13:03:00;0,0,0,0,0,0,0,0,0,0;
T5;S3d2;12:23:00;0,0,0,0,0,0,0,0,0,0;
T15;S3d2;10:43:00;0,0,0,0,0,0,0,0,0,0;

5.3.3 Problemas encontrados

A continuación se resumen brevemente algunos de los problemas que se encontraron durante la implementación final del proyecto:

- Comunicación Servicio-Actividad: Uno de los problemas encontrados era la comunicación de la clase servicio con la clase actividad ya que había variables que debían ser usadas en ambas, este problema se solucionó utilizando las Shared Preferences de Android, el cual no deja de ser un fichero XML donde se guardan el valor de las variables establecidas como Shared Preferences. Esta solución además de permitir usar la misma variable en ambas clases solucionaba otro problema, el cual era dar almacenar las preferencias de los usuarios para que en la próxima ejecución el usuario no tuviera que configurar estas preferencias.

- Gráficas: Otro de los problemas encontrados era como obtener los datos de las gráficas debido a que la aplicación del dispositivo móvil no disponía de todos los datos de los sensores (porque la aplicación podría haber estado parada o sin la cobertura suficiente para recibir todos los valores) de forma que era necesario obtener de alguna forma los últimos 10 valores de los sensores de la parte servidora (la cual si dispone siempre de estos valores), utilizando el mismo protocolo MQTT para obtener estos datos siendo el servidor el que genera los valores y publicándolos al mismo tiempo al dispositivo móvil.

Capítulo 6

PRUEBAS

En este capítulo se van a presentar las pruebas realizadas al sistema, con las cuales aseguraremos que los requisitos definidos al principio del proyecto se han cumplido. En un primer punto se describirá el entorno de pruebas donde se han realizado, para continuar en los siguientes puntos con las pruebas llevadas a cabo.

6.1 Descripción entorno de pruebas

El entorno de pruebas a esta formado por dos partes por un lado la parte cliente formado por un dispositivo Android con la aplicación desarrollada instalada y por otro lado la parte servidora que se ha tenido que crear para poder hacer las pruebas.

6.1.1 Parte cliente

Las pruebas se han desarrollado sobre varios dispositivos Android que a continuación se detallan sus características

HTC Wildfire:

- SO: Android 2.2
- Memoria: 384MB RAM
- CPU: MSM 7225 528Mhz
- GPS: Sí
- Dimensiones: 106,75x60,4x12,9 mm
- Peso: 118 gr.
- Pantalla: Capacitiva
- Resolución pantalla: 320x240 pixels
- Otros sensores: sensor ambiental, acelerómetro

Ilustración 53: HTC Wildfire

HTC Sensation:

- SO: Android 4.0
- Memoria: 768MB RAM
- CPU: dualcore 1,2 Ghz
- GPS: Sí, A-GPS
- Dimensiones: 126,1x65,4x11,3 mm
- Peso: 148 gr.
- Pantalla: Capacitiva
- Resolución pantalla: 960x540 pixels

- Otros sensores: sensor proximidad, acelerómetro

Ilustración 54: HTC Sensation

Nexus 4:

- SO: Android 4.2
- Memoria: 2GB RAM
- CPU: quad-core Qualcomm Snapdragon APQ8064 1.5 GHz
- GPS: Sí, GPS con soporte A-GPS; GLONASS
- Dimensiones: 133.9 x 68.7 x 9.1 mm
- Peso: 139 gr.
- Pantalla: Capacitiva
- Resolución pantalla: 768 x 1280 pixels
- Otros sensores: sensor proximidad, acelerómetro, etc..

Ilustración 55: Nexus 4

6.1.2 Parte servidora

Uno de los elementos más importantes del *backend* es la red de sensores, debido que el objetivo del proyecto no era la de desarrollar una red de sensores ni la integración de esta con la parte con la parte servidora se ha emulado la red de sensores con un programa desarrollado en JAVA, el cual saca los datos de los sensores de una BBDD MySQL, publicando estos datos en el *bróker* MQTT para que los dispositivos móviles que estén suscritos puedan obtener los datos.

Cabe destacar que los datos incluidos en la BBDD MySQL son datos reales obtenidos en un anterior Proyecto en el cual la red de sensores estaba formado por los siguientes dispositivos:

Base receptora:

Ilustración 56: Base receptora

Sensor presión:

Ilustración 57: Sensor Presión

Sensor corriente:

Ilustración 58: Sensor corriente

Sensor movimiento:

Ilustración 59: sensor movimiento

Además de la BBDD en MySQL y la aplicación JAVA encargada de publicar los datos de los sensores en el servidor se encuentra el bróker MQTT que es el encargado de gestionar toda la comunicación con los dispositivos móviles, el *bróker* MQTT utilizado se llama Really Small Message Broker y está desarrollado por IBM

6.2 Validación sistema

6.2.1 Casos de prueba

Los *Test Case* o casos de prueba son un conjunto de variables o condiciones gracias a las cuáles podrá determinar el analista si lo especificado en el apartado del análisis es parcial o completamente satisfactorio. Para ello se ha decidido realizar un caso de prueba por cada uno de los casos de uso expuestos en el apartado 5.1.4

En la descripción textual de cada uno de los casos de prueba, se especificarán los campos que se muestran a continuación:

- Identificador: Es un identificador único para futuras referencias, por ejemplo, mientras se describe un defecto encontrado.

- Propósito: Contiene una breve descripción del propósito de la prueba, y la funcionalidad que chequea.

- Configuración: Contiene información acerca de la configuración del hardware o software en el cuál se ejecutará el caso de prueba.

- Inicialización: Describe acciones, que deben ser ejecutadas antes de que los casos de prueba se hayan inicializado. Por ejemplo, debemos abrir algún archivo.

- Finalización: Describe acciones, que deben ser ejecutadas después de realizado el caso de prueba. Por ejemplo si el caso de prueba estropea la

base de datos, el analista debe restaurarla antes de que otro caso de prueba sea ejecutado.

- Acciones: Pasos a realizar para completar la prueba.

- Salida esperada: Contiene una descripción de lo que el analista debería ver tras haber completado todos los pasos de la prueba.

- Resultado: Indica el resultado cualitativo de la ejecución del caso de prueba, a menudo con un Correcto/Fallido.

- Severidad: Indica el impacto del defecto en el sistema: Grave, Mayor, Normal, Menor.

A continuación se muestran los casos de prueba:

ID: CP-01	
Propósito:	Mostrar el listado de los sensores
Configuración:	El *host* y el *topic* deben estar correctamente configurados y el dispositivo debe tener conectividad
Inicialización:	Abrir la aplicación
Finalización:	No aplica
Acciones:	1. Esperar que se abra la aplicación 2. Publicar datos de un sensor concreto desde el servidor 3. Comprobar los datos mostrados
Salida esperada:	Cuando la aplicación ha sido ejecutada por primera vez, la lista debe ser vacía, una vez que la aplicación se ha ejecutado y se han publicado deberá mostrar la lista con el sensor concreto
Resultado:	Correcto
Severidad:	Grave

Tabla 42: Caso de prueba CP-01

ID: CP-02	
Propósito:	Establecer conexión con el servidor y empezar a recibir datos

Configuración:	El *host* y el *topic* deben estar correctamente configurados y el dispositivo debe tener conectividad, tener habilitado la notificaciones
Inicialización:	Abrir la aplicación
Finalización:	No aplica
Acciones:	1. Esperar que se establezca la conexión (se muestra un *pop-up* cuando se establece) 2. Publicar un dato de un sensor 3. Comprobar la correcta recepción del evento
Salida esperada:	Recibir en el dispositivo una notificación con el evento publicado en el servidor
Resultado:	Correcto
Severidad:	Grave

Tabla 43: Caso de prueba CP-02

ID: CP-03	
Propósito:	Ver la gráfica de un sensor elegido sobre la lista de sensores
Configuración:	El *host* y el *topic* deben estar correctamente configurados y el dispositivo debe tener conectividad.
Inicialización:	Abrir la aplicación y establecer la conexión
Finalización:	No aplica
Acciones:	1. Esperar a que se muestre la lista de sensores disponibles 2. Publicar los datos de grafica de un sensor 3. Seleccionar dicho sensor 4. Comprobar la gráfica mostrada con la publicada
Salida esperada:	La grafica con los datos publicados
Resultado:	Correcto
Severidad:	Mayor

Tabla 44: Caso de prueba CP-03

ID: CP-04	
Propósito:	Editar las preferencias de la aplicación
Configuración:	No necesita ninguna configuración
Inicialización:	Abrir la aplicación
Finalización:	No aplica
Acciones:	1. Apretar el botón de *Menú* 2. Elegir *preferences* 3. Editar una de las preferencias 4. Volver atrás

	5. Salir de la aplicación 6. Volver a entrar en la aplicación y en el menú de preferencias 7. Comprobar el valor de la preferencia editada
Salida esperada:	La preferencia con el valor introducido anteriormente
Resultado:	Correcto
Severidad:	Mayor

Tabla 45: Caso de prueba CP-04

ID: CP-05	
Propósito:	Minimizar la aplicación y comprobar que se siguen recibiendo las notificaciones
Configuración:	El *host* y el *topic* deben estar correctamente configurados y el dispositivo debe tener conectividad, tener habilitado la notificaciones
Inicialización:	Abrir la aplicación
Finalización:	No aplica
Acciones:	1. Esperar que se establezca la conexión (se muestra un pop-up cuando se establece) 2. Apretar el botón de *Menú* 3. Elegir *Home* 4. Publicar datos de sensores en el servidor
Salida esperada:	Notificaciones con los datos de los sensores
Resultado:	Correcto
Severidad:	Mayor

Tabla 46: Caso de prueba CP-05

ID: CP-06	
Propósito:	Salir de la aplicación y comprobar que aunque se tienen habilitadas las notificaciones no se reciben
Configuración:	El *host* y el *topic* deben estar correctamente configurados y el dispositivo debe tener conectividad, tener habilitado la notificaciones
Inicialización:	Abrir la aplicación
Finalización:	No aplica
Acciones:	1. Esperar que se establezca la conexión (se muestra un *pop-up* cuando se establece) 2. Apretar el botón de Menú 3. Elegir *Exit* 4. Publicar datos de sensores en el servidor

Salida esperada:	Ninguna notificación
Resultado:	Correcto
Severidad:	Mayor

Tabla 47: Caso de prueba CP-06

6.2.2 Evaluación rendimiento

En este capítulo se evalúa el rendimiento de la implementación a través de una serie de experimentos para determinar si la naturaleza dinámica del sistema, resulta en penalidades de cara al rendimiento.

Para estos experimentos se empleó un hardware dedicado, tal y como se mencionó anteriormente, se han empleado *smartphones* Android para testear los clientes móviles, ya que este tipo de dispositivos puede ser fácilmente integrable con otro software Java. El ancho de banda bruto del enlace inalámbrico entre los clientes móviles y el sistema de publicación / suscripción se garantizó que era estable durante la evaluación.

La Ilustración 59 muestra el impacto del número de editores (en el ámbito de este Proyecto serían los sensores de la red) en el rendimiento de los mensajes (*throughput*). En este caso el experimento se llevó a cabo con un número creciente de editores y con un único suscriptor.

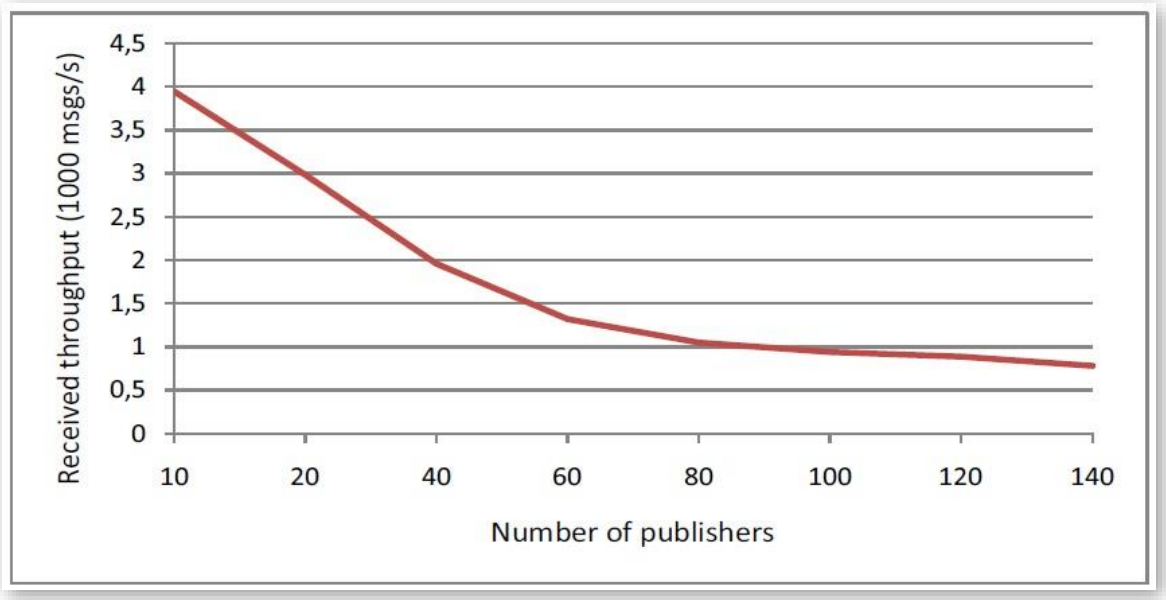


Ilustración 60: Impacto entre editores y rendimiento de mensajes

El sistema soporta una producción de 4000 mensajes por segundo para unos pocos editores y sobre 450 msgs/s para una cantidad de 140 editores.

De la misma también se comprobó el impacto del número de suscriptores. Ambos experimentos se repitieron al menos cinco veces. Tal y como muestra la siguiente ilustración, la tasa de recepción de mensajes disminuye de forma significativa con un creciente número de suscriptores.

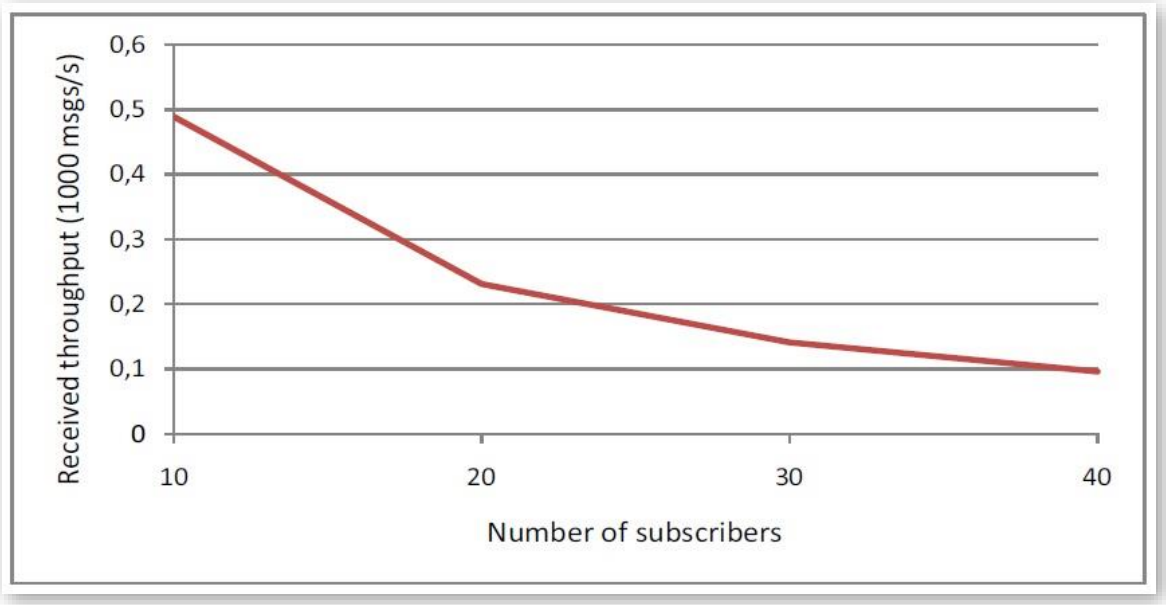


Ilustración 61: Impacto entre suscriptores y rendimiento de mensajes

Por último se comprobó la latencia o el tiempo requerido por un cliente en recibir un flujo de datos de un único editor, como es obvio, en una red *wireless*. La latencia es una importante métrica a considerar, especialmente donde el intercambio de datos en *soft real-time* es un requisito. Dicha gráfica sobre la latencia puede apreciarse en la Ilustración 61.

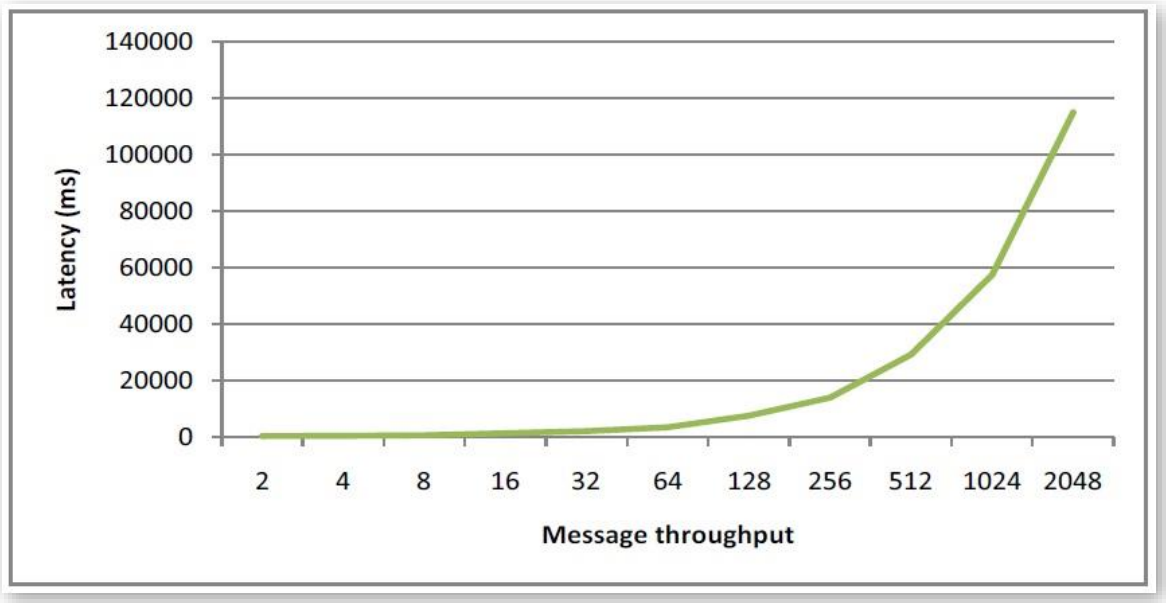


Ilustración 62: Tiempo de latencia para el cliente remoto MQTT. El eje x en escala logarítmica muestra el número de eventos

Capítulo 7

CONCLUSIONES

En este Proyecto Fin de Carrera se ha creado para el sistema operativo Android una aplicación enfocada en el ámbito de la telemonitorización, la cual consiste en la recepción de mensajes publicados por una red de sensores, los mensajes están compuestos por el estado del sensor en un momento dado en tiempo real mediante notificaciones *push*. La función de la aplicación es la de recibir los datos para monitorizar el flujo de actividades de la red de sensores, mostrar la información y almacenarla de forma persistente para un posible uso a posteriori.

Para la implementación de la aplicación ha sido necesario estudiar el desarrollo de aplicaciones en Android ya que aunque el lenguaje empleado ha sido Java el universo de Android está compuesto por clases propias y por un ciclo de vida propio para las aplicaciones. Además del estudio del ciclo de vida de las aplicaciones de Android ha sido necesario el estudio en profundidad de protocolos de comunicación, en concreto de MQTT basado en el modelo *publish/subscribe*, necesario para poder establecer conexiones *push*, consiguiendo así uno de los objetivos primordiales del proyecto.

Gracias a las pruebas realizadas se ha comprobado que quedan cumplidas las expectativas que se vieron reflejadas en los requisitos recogidos en la fase de análisis.

Más allá de lo exigido por el cliente y del entorno concreto para el que se ha desarrollado, esta aplicación sirve de base sólida con unas ligeras modificaciones para cualquier sistema de telemonitorización básico que se plantee. Por último se ha tenido en cuenta el hecho de que los receptores iban a ser dispositivos móviles, procurando promover el mínimo consumo posible de recursos de cara a no drenar la vida de la batería (no realizar transacciones innecesarias, empleo de una base de datos ligera como SQLite, etc.).

Analizando los resultados del rendimiento de las pruebas realizadas se comprueba que el rendimiento del sistema es más que suficiente para el dimensionamiento de la red de sensores a las que está orientado este proyecto. La utilización del protocolo MQTT, en detrimento de otro protocolo como C2DM, ha sido un acierto ya que además de proporcionar un gran rendimiento (hasta 4000 mensajes por segundos) permite que la aplicación tenga más líneas futuras debido a sus capacidades de tener una comunicación bidireccional, característica que no contempla C2DM.

El sistema completo está desarrollado sobre el patrón arquitectónico del Modelo Vista Controlador o MVC, permitiendo separar la lógica del sistema, del acceso a los datos y de las interfaces de usuario. Esto es especialmente útil para fomentar la extensibilidad y reutilización en un sistema que de seguro seguirá aumentando en funcionalidad y características en un futuro.

A lo largo de la vida del proyecto se solventaron diversos problemas hasta dar con la aplicación funcional que se tiene actualmente. Uno de los problemas al que se tuvo que hacer frente es a la forma de almacenar las preferencias de los usuarios para poder ser compartida entre la “Actividad” y el “Servicio” así como poder almacenar de forma persistente los mensajes recibidos por la aplicación, la solución fue utilizar métodos nativos de Android desarrollados para estos cometidos demostrando ser así Android un sistema maduro.

Capítulo 8

LÍNEAS FUTURAS

En este capítulo se planten líneas de desarrollo que pueden ser estudiadas y llevadas a cabo en un futuro. Entre estas futuras líneas, se proponen las siguientes:

- Mejoras en la interfaz de usuario: La aplicación ha sido realizada mediante la versión de Android 2.1, en los últimos tiempos Android ha ido mejorando el aspecto visual sobre todo a partir del tema *Holo* que se implementó a partir de la versión 4.0 por lo que para un futuro sería recomendable actualizar la aplicación con una nueva interfaz

- Multiplataforma: Android es uno de los sistemas operativos más utilizados en los dispositivos móviles pero no el único por lo que sería recomendable en un futuro ofrecer la aplicación como multiplataforma ofreciendo la aplicación a sistemas consolidados como IOS o a otros con llegan con fuerza como pueda ser Windows Phone.

- Mejoras en las gráficas: Actualmente las gráficas se "dibujan" usando funciones nativas de Android pero existen proyectos de código abierto, como achartengine, los cuales proporcionan librerías para Android para generar unas gráficas mejoradas pudiendo ser este un punto a mejorar a futuro en la aplicación.

- Idioma: Sería interesante generar la aplicación en un mayor número de idiomas y añadir la selección del idioma en las preferencias de la aplicación.

- Actuador: El único objetivo actual de la aplicación es recibir los datos publicados por los sensores, en un futuro se podría implementar una nueva funcionalidad a la aplicación, la cual sería de funcionar como actuador sobre los sensores pudiendo modificar su configuración o comportamiento desde el dispositivo móvil (en vez de solo recibir mensajes también sería capaz de publicar mensajes)

BIBLIOGRAFIA

[1] IBM Simon
http://en.wikipedia.org/wiki/IBM_Simon

[2] Iphone
http://en.wikipedia.org/wiki/IPhone

[3] Nielsen report
http://www.nielsen.com/global/en.html

[4] Google compra Android Inc.
http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

[5] Anuncio G1.
http://es.t-mobile.com/company/PressReleases_Article.aspx?assetName=Prs_Prs_20080923

[6] Vida de Android.
http://www.droidlife.com/

[7] App Runner
http://www.netmite.com/android/

[8] Formatos soportados por Android
http://developer.android.com/guide/appendix/media-formats.html

[9] Usar Android como modem Wifi
http://developer.android.com/resources/articles/speech-input.html

[10] Anuncion Android 1.0 SDK, release 1
http://android-developers.blogspot.com.es/2008/09/announcing-android-10-sdk-release-1.html

[11] Android 1.1 SDK, disponible
http://android-developers.blogspot.com.es/2009/02/android-11-sdk-release-1-now-available.html

[12] Android Distribución versiones
http://developer.android.com/about/dashboards/index.html

[13] Cuotas de Mercado SOs móviles

http://windowsphoneapps.es/2013/01/windows-phone-sigue-aumentando-su-cuota-de-mercado/

[14] Comparativa SOs móviles
http://planetamoviles.com/descubre-las-principales-diferencias-entre-los-sistemas-jelly-bean-ios-6-y-windows-phone-8/

[15] Polling notifications
http://josecortes.net/blog/2011/10/16/push-notifications-vs-polling/

[16] Cugola, G., Murphy, A.L., Picco
Content-Based Publish-subscribe in a Mobile Environment.
Bellavista, P., Corradi, A. (eds.) Mobile Middleware, pp. 257–285. Auerbach Publications (2006), invited contribution

[17] Shnayder, V., Chen, B.r., Lorincz, K., Jones, T.R.F.F., Welsh, M.: Sensor networks for medical care. In: Proceedings of the 3rd international conference on Embedded networked sensor systems. pp. 314–314.

[18] Fiege, L., Grtner, F.C., Kasten, O., Zeidler, A.:
Supporting mobility in contentbased *publish/subscribe* middleware (2003)

[19] Patrón *publish/subscribe*
http://gpec2010.googlecode.com/svn/trunk/docs/_build/html/tecnologias.html

[20] Van de Laar, F. (2002). *Publish/subscribe* as architectural style for component interaction (Mater's thesis), Phillips Research Laboratories, Eindhoven

[21] Android Cloud to Device Messaging Framework
https://developers.google.com/android/c2dm

[22] MQTT protocol
http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[23] Eclipse
http://es.wikipedia.org/wiki/Eclipse_(software)

[24] Android SDK
http://developer.android.com/sdk/exploring.html

[25] OEM USB Drivers
http://developer.android.com/tools/extras/oem-usb.html#Drivers

[26] Plugin eclipse
http://developer.android.com/tools/sdk/eclipse-adt.html

[27] Mysql
http://www.mysql.com/products/

[28] Java

http://java.com/es/download/faq/whatis_java.xml

[29] Sensordrone
http://www.kickstarter.com/projects/453951341/sensordrone-the-6th-sense-of-your-*smartphone*and-be

[30] iMon
http://www.cctvcentersl.es/aldia/ad2011/ad0111_CENTER-IV/iMON_Android_esp.pdf

[31] mHealthalert
http://www.mhealthalert.com/index.php/en/mhealth-alert-3

[32] Glucometro
https://play.google.com/store/apps/details?id=com.fjbelchi.glucosemeter2&feature=related_apps

[33] Waspmote
http://www.libelium.com/es/products/waspmote/

[34] GMail
https://play.google.com/store/apps/details?id=com.google.android.gm&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5nb29nbGUuYW5kcm9pZC5nbSJd

[35] Ingeniería del Software.
http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software

[36] Agencia Espacial Internacional.
http://www.esa.int/esaCP/index.html

[37] Estándar de Ingeniería del Software de ESA.
ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS050.pdf

[38] Winston W. Royce (1970). Managing the development of large software systems.
http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

[39] Matriz de Trazabilidad.
http://www.softqatest.net/?p=77

Apéndice A

MANUAL DE USUARIO

Este anexo se presenta un breve tutorial sobre el manejo de la aplicación, en él se incluyen instrucciones de cómo ejecutarla y configurar los parámetros de la misma.

- Instalación

  Antes de poder realizar la instalación de la aplicación es necesario configurar el dispositivo para que pueda realizarse la instalación de aplicaciones de fuentes desconocidas, por eso habilitaremos la opción “Fuentes desconocidas” ubicado en la parte de seguridad de ajustes del dispositivo.

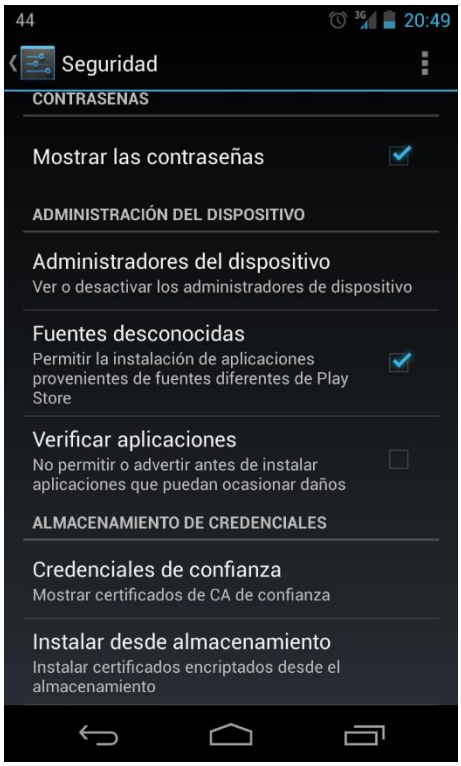


Ilustración 63: Ajustes -> Seguridad

Una vez modificado la configuración debemos abrir un explorador para poder buscar el apk de la aplicación, en este caso se ha utilizado *dropbox*

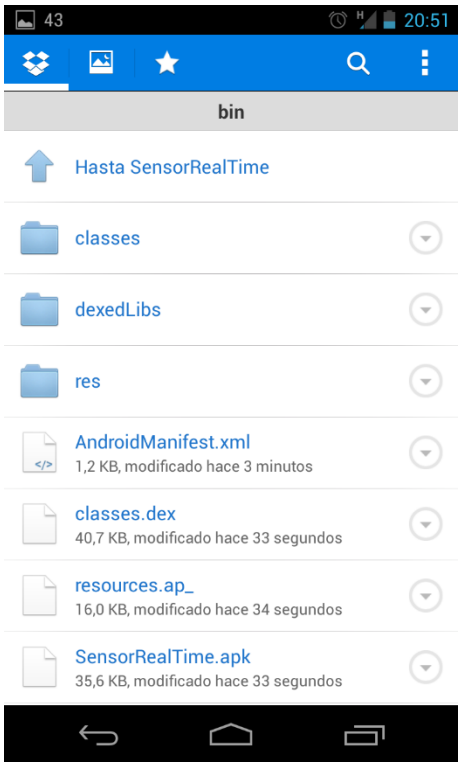


Ilustración 64: Explorador Dropbox

Seleccionamos la apk, SensorRealTime.apk, y ejecutamos el archivo, nos saldrá una ventana con los permisos que necesita la aplicación, seleccionaremos "Instalar"

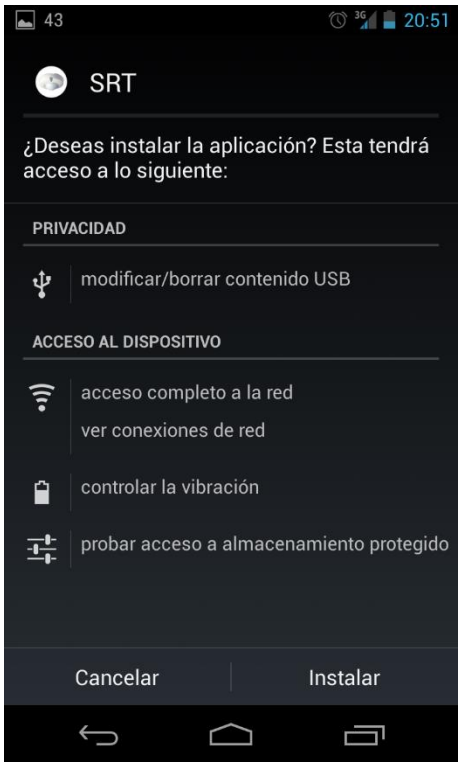


Ilustración 65: Ventana de instalación Android

Después de un periodo de tiempo en el que se realiza la instalación, se muestra una pantalla indicando que la instalación se ha realizado correctamente.

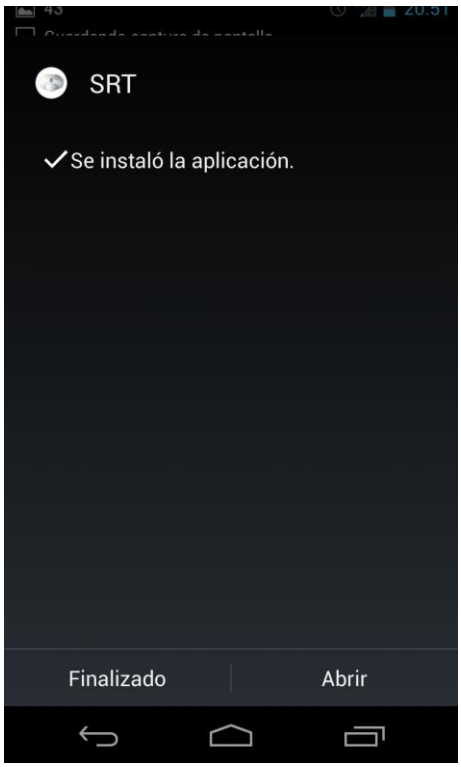


Ilustración 66: Ventana finalización instalación

Una vez terminada la instalación podemos comprobar que la aplicación ya se encuentra en el menú con el resto de aplicaciones

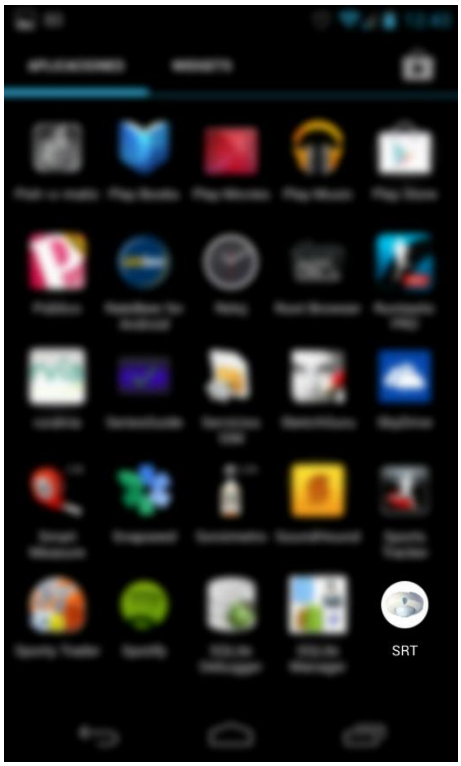


Ilustración 67: Menú de aplicaciones

- Configuración

  Una vez ejecutada la aplicación podemos realizar la configuración de la misma, desde el menú principal podemos mostrar un desplegable con las opciones pulsando el botón de Menú del dispositivo. Se mostraran las opciones (*Home*, *Preferences* y *Exit*)

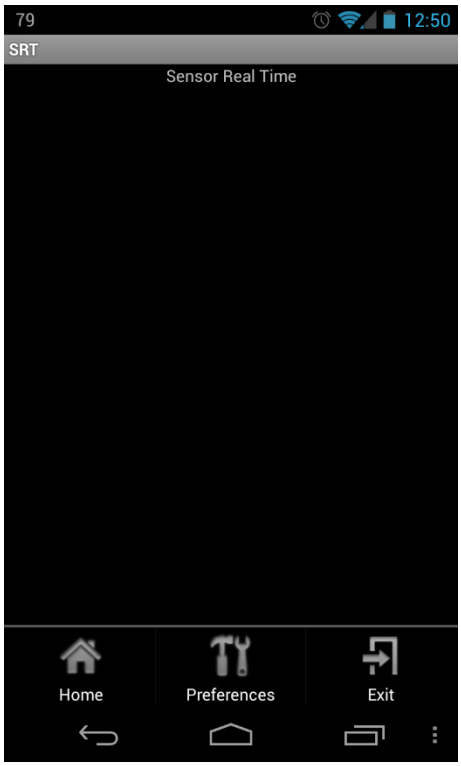


Ilustración 68: Menú aplicación

Pulsando la opción de "*Preferences*" podremos configurar la aplicación a nuestro gusto.

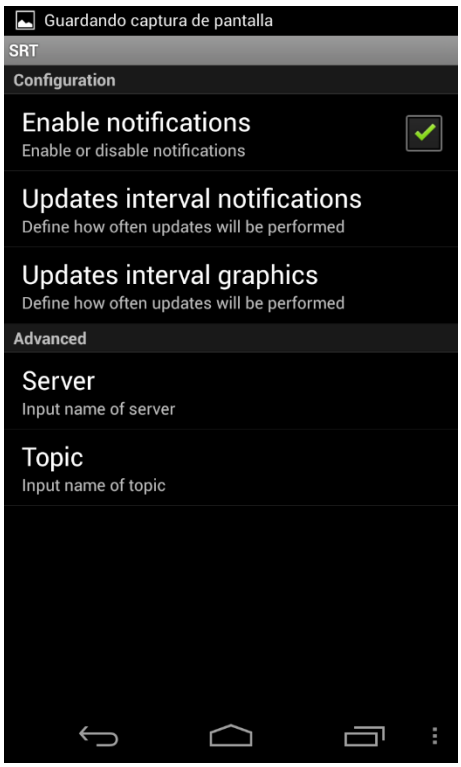


Ilustración 69: Menú Preferencias

Editar el intervalo de las gráficas

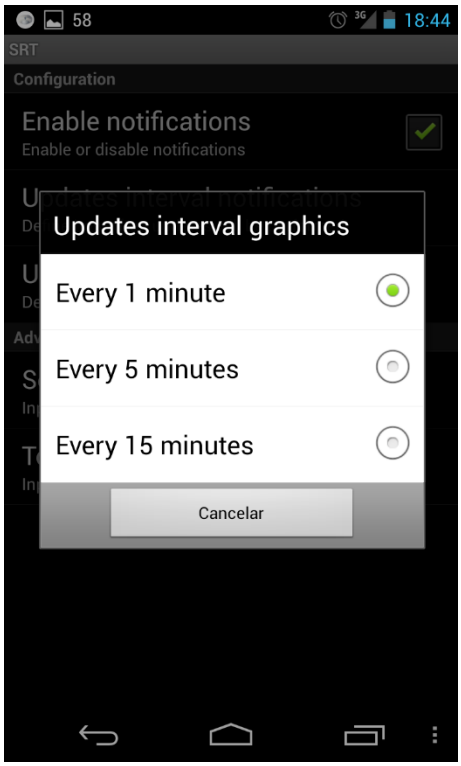


Ilustración 70: Editar intervalo gráficas

Editar el servidor

Ilustración 71: Editar servidor

Editar el nombre del *topic*

Ilustración 72: Editar nombre topic

- Funcionamiento

  Cuando ejecutamos por primera vez la aplicación nos encontraremos que la aplicación no muestra ningún dato. Cada vez que ejecutamos la aplicación se muestra un *pop-up* cunado se establece la conexión al servidor, indicando el servidor y en otro *pop-up* aparte el *topic*

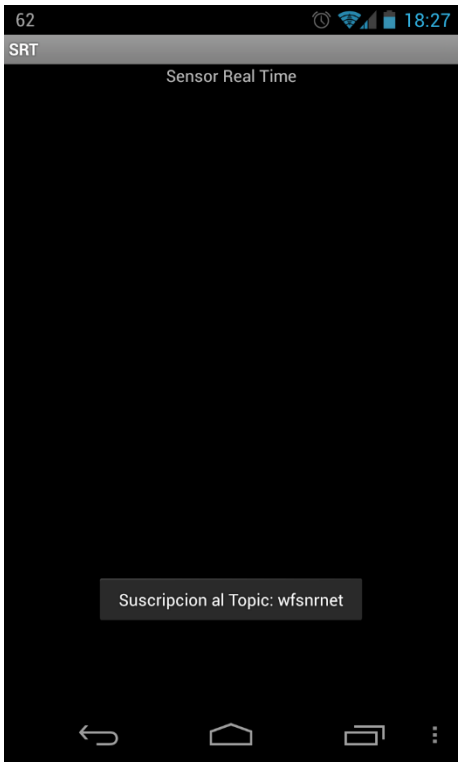


Ilustración 73: ***Pop-up topic***

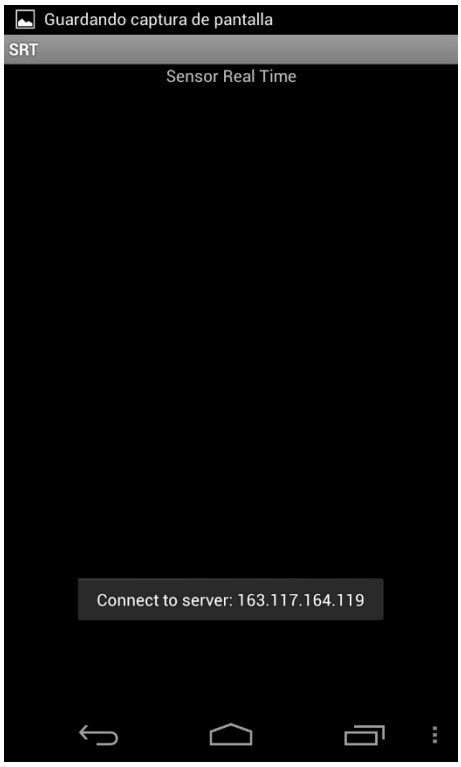


Ilustración 74: ***Pop-up*** **servidor**

Una vez realizada la conexión la aplicación empezara a recibir los datos de los sensores y si están habilitadas las notificaciones se mostraran estas.

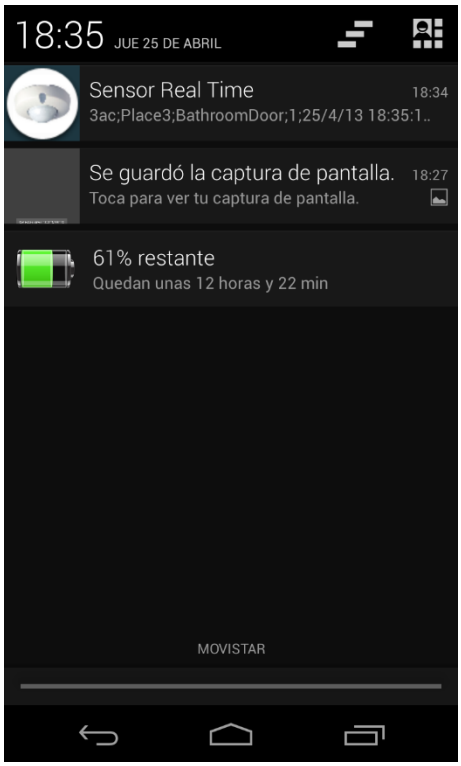


Ilustración 75: Notificación aplicación

Según se vayan recibiendo las notificaciones la pantalla de inicio de la aplicación ira mostrando los últimos datos recibidos del servidor

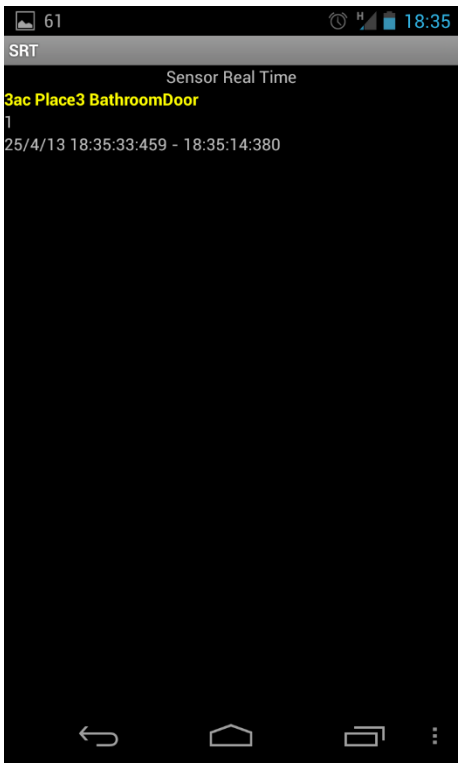


Ilustración 76: Pantalla inicial con el dato de un sensor

Según se vayan recibiendo datos de distintos sensores la lista de sensores de la pantalla inicial crecerá dinámicamente

Ilustración 77: Lista de sensores dinámica

Seleccionando a un sensor, se mostrará la gráfica de dicho sensor

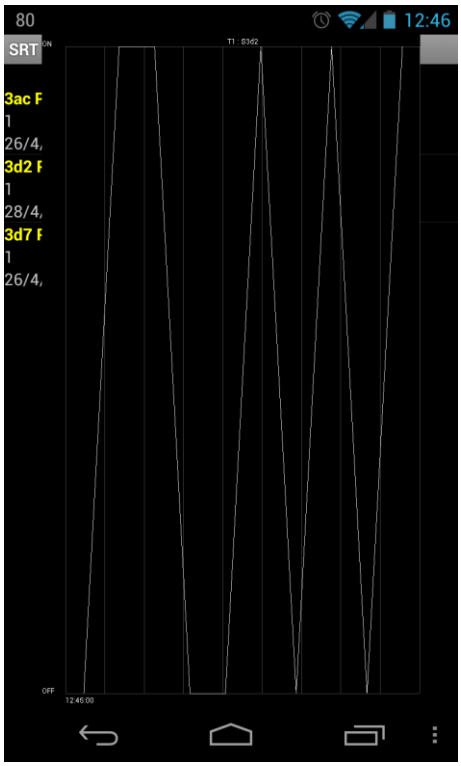


Ilustración 78: Gráfica sensor

Apéndice B

PLANIFICACIÓN

En este apéndice del proyecto se muestra la planificación que se ha ido realizando durante el desarrollo de la aplicación. Para ello se utiliza el Diagrama de Gantt, el cual muestra el tiempo de dedicación previsto y real para las diferentes tareas o actividades a lo largo del tiempo de desarrollo.

Para analizar la planificación seguida, se van a mostrar tres diagramas diferentes:

- En la ilustración 79 se muestra la planificación ideal cuando se inició el proyecto.

- En la ilustración 80 se muestra el tiempo real de dedicación en el desarrollo del sistema.

- En las ilustraciones 81 y 82 se muestra una comparativa entre la planificación inicial y el tiempo real empleado en el desarrollo, mostrándose la desviación temporal existente.

Tal y como se ve en los diagramas de Gantt, ha habido distintas desviaciones en las tareas previamente definidas aunque el tiempo total del proyecto se ha mantenido, dentro de estas desviación cabe destacar el estudio previo el cual estaba planificado en 5 días y se prolongó hasta los 15 días debido a la investigación de métodos para realizar *push* en la aplicación y también se puede comprobar que el periodo de análisis y diseño

de la aplicación ha sido menor de los planificados, debido a la simplicidad de la aplicación y a la utilización del protocolo MQTT, este tiempo ganado ha sido posteriormente compensado por la demora en la fase de implementación debido a la inexperiencia en desarrollo de aplicaciones en Android..

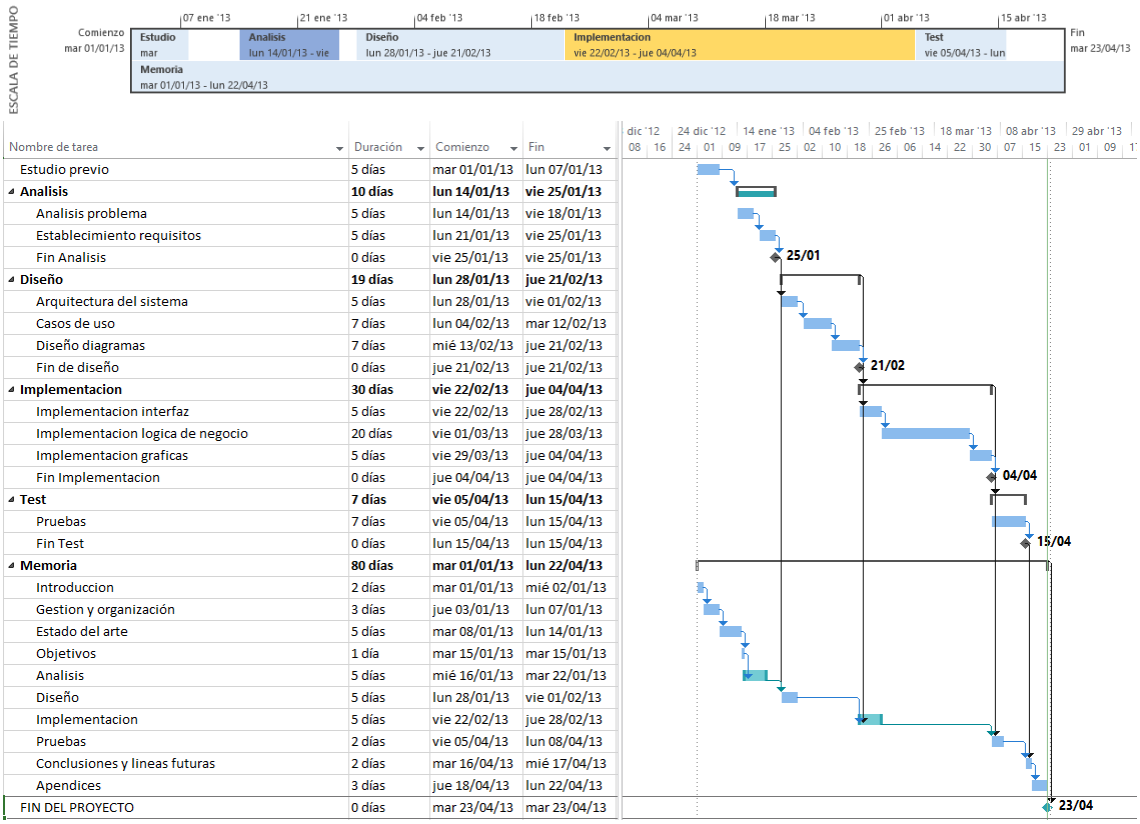
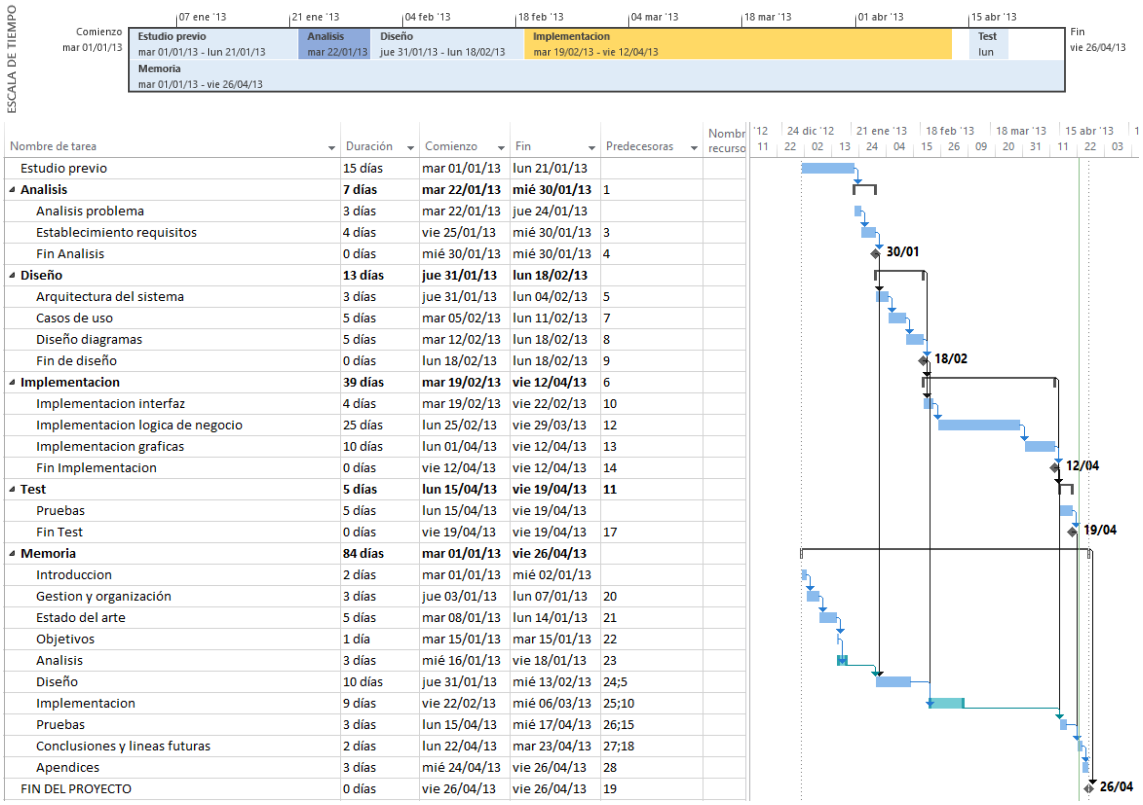


Ilustración 79: Planificación inicial



Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombr recurso
Estudio previo	15 días	mar 01/01/13	lun 21/01/13		
Analisis	**7 días**	**mar 22/01/13**	**mié 30/01/13**	1	
Analisis problema	3 días	mar 22/01/13	jue 24/01/13		
Establecimiento requisitos	4 días	vie 25/01/13	mié 30/01/13	3	
Fin Analisis	0 días	mié 30/01/13	mié 30/01/13	4	
Diseño	**13 días**	**jue 31/01/13**	**lun 18/02/13**		
Arquitectura del sistema	3 días	jue 31/01/13	lun 04/02/13	5	
Casos de uso	5 días	mar 05/02/13	lun 11/02/13	7	
Diseño diagramas	5 días	mar 12/02/13	lun 18/02/13	8	
Fin de diseño	0 días	lun 18/02/13	lun 18/02/13	9	
Implementacion	**39 días**	**mar 19/02/13**	**vie 12/04/13**	6	
Implementacion interfaz	4 días	mar 19/02/13	vie 22/02/13	10	
Implementacion logica de negocio	25 días	lun 25/02/13	vie 29/03/13	12	
Implementacion graficas	10 días	lun 01/04/13	vie 12/04/13	13	
Fin Implementacion	0 días	vie 12/04/13	vie 12/04/13	14	
Test	**5 días**	**lun 15/04/13**	**vie 19/04/13**	**11**	
Pruebas	5 días	lun 15/04/13	vie 19/04/13		
Fin Test	0 días	vie 19/04/13	vie 19/04/13	17	
Memoria	**84 días**	**mar 01/01/13**	**vie 26/04/13**		
Introduccion	2 días	mar 01/01/13	mié 02/01/13		
Gestion y organización	3 días	jue 03/01/13	lun 07/01/13	20	
Estado del arte	5 días	mar 08/01/13	lun 14/01/13	21	
Objetivos	1 día	mar 15/01/13	mar 15/01/13	22	
Analisis	3 días	mié 16/01/13	vie 18/01/13	23	
Diseño	10 días	jue 31/01/13	mié 13/02/13	24;5	
Implementacion	9 días	vie 22/02/13	mié 06/03/13	25;10	
Pruebas	3 días	lun 15/04/13	mié 17/04/13	26;15	
Conclusiones y lineas futuras	2 días	lun 22/04/13	mar 23/04/13	27;18	
Apendices	3 días	mié 24/04/13	vie 26/04/13	28	
FIN DEL PROYECTO	0 días	vie 26/04/13	vie 26/04/13	19	

Ilustración 80: Planificación Final

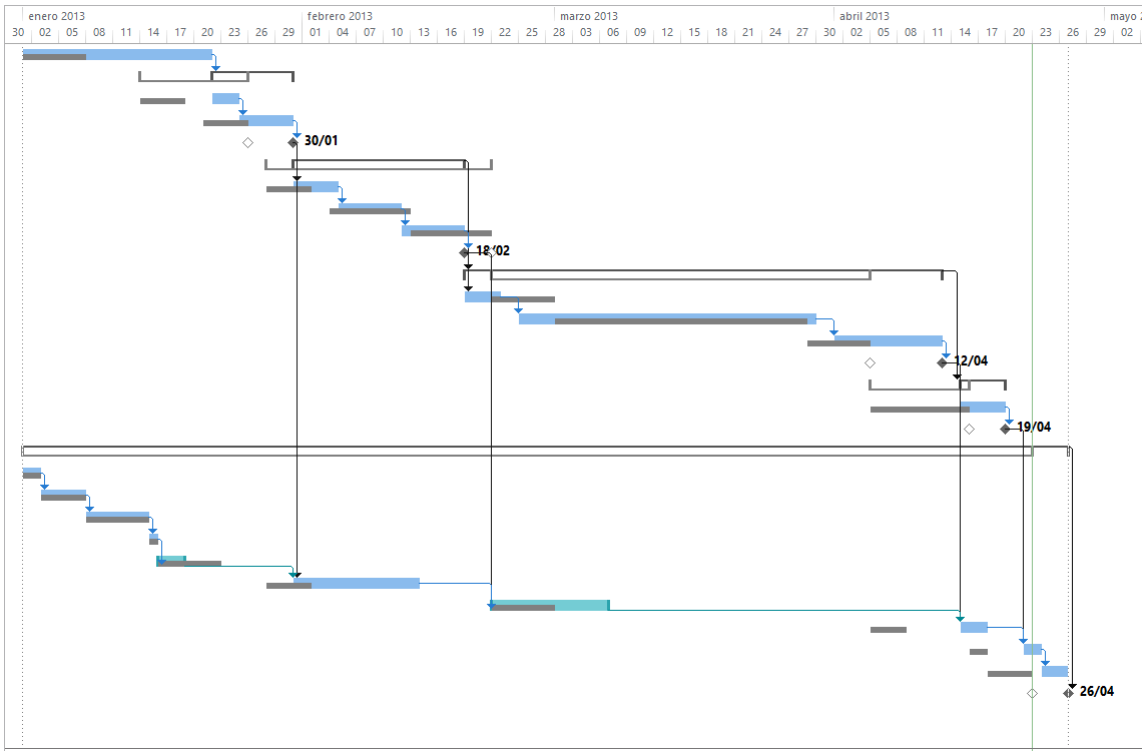


Ilustración 81: Comparativa entre planificaciones (I)

Nombre de tarea	Duración	Comienzo	Fin	Variación de duración
Estudio previo	15 días	mar 01/01/13	lun 21/01/13	10 días
Analisis	**7 días**	**mar 22/01/13**	**mié 30/01/13**	**-3 días**
Analisis problema	3 días	mar 22/01/13	jue 24/01/13	-2 días
Establecimiento requisitos	4 días	vie 25/01/13	mié 30/01/13	-1 día
Fin Analisis	0 días	mié 30/01/13	mié 30/01/13	0 días
Diseño	**13 días**	**jue 31/01/13**	**lun 18/02/13**	**-6 días**
Arquitectura del sistema	3 días	jue 31/01/13	lun 04/02/13	-2 días
Casos de uso	5 días	mar 05/02/13	lun 11/02/13	-2 días
Diseño diagramas	5 días	mar 12/02/13	lun 18/02/13	-2 días
Fin de diseño	0 días	lun 18/02/13	lun 18/02/13	0 días
Implementacion	**39 días**	**mar 19/02/13**	**vie 12/04/13**	**9 días**
Implementacion interfaz	4 días	mar 19/02/13	vie 22/02/13	-1 día
Implementacion logica de negocio	25 días	lun 25/02/13	vie 29/03/13	5 días
Implementacion graficas	10 días	lun 01/04/13	vie 12/04/13	5 días
Fin Implementacion	0 días	vie 12/04/13	vie 12/04/13	0 días
Test	**5 días**	**lun 15/04/13**	**vie 19/04/13**	**-2 días**
Pruebas	5 días	lun 15/04/13	vie 19/04/13	-2 días
Fin Test	0 días	vie 19/04/13	vie 19/04/13	0 días
Memoria	**84 días**	**mar 01/01/13**	**vie 26/04/13**	**4 días**
Introduccion	2 días	mar 01/01/13	mié 02/01/13	0 días
Gestion y organización	3 días	jue 03/01/13	lun 07/01/13	0 días
Estado del arte	5 días	mar 08/01/13	lun 14/01/13	0 días
Objetivos	1 día	mar 15/01/13	mar 15/01/13	0 días
Analisis	3 días	mié 16/01/13	vie 18/01/13	-2 días
Diseño	10 días	jue 31/01/13	mié 13/02/13	5 días
Implementacion	9 días	vie 22/02/13	mié 06/03/13	4 días
Pruebas	3 días	lun 15/04/13	mié 17/04/13	1 día
Conclusiones y lineas futuras	2 días	lun 22/04/13	mar 23/04/13	0 días
Apendices	3 días	mié 24/04/13	vie 26/04/13	0 días
FIN DEL PROYECTO	0 días	vie 26/04/13	vie 26/04/13	0 días

Ilustración 82: Comparativa entre planificaciones (II)

Apéndice C

PRESUPUESTO

En este apéndice del proyecto se muestra el presupuesto para el desarrollo del sistema, en el que se hace una descripción detallada de los gastos del proyecto, incluyendo el personal que ha desarrollado el sistema y los equipos utilizados para ello. El presupuesto detallado se encuentra en la siguiente ilustración

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Isidro Muñoz Sanchez
2.- Departamento:
UC3M
3.- Descripción del Proyecto:

- Titulo	**Monitorización de redes de sensores inalámbricas a través de dispositivos Android**	
- Duración (meses)	**4 Meses**	
Tasa de costes Indirectos:		**20%**

4.- Presupuesto total del Proyecto (valores en Euros):
9.563,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	Categoría	Horas	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Jefe de proyecto	Jefe de proyecto	224,00	1,71	2.500,00	4.266,67	
Analista	Analista	108,00	0,82	2.000,00	1.645,71	
Programador	Programador	156,00	1,19	1.500,00	1.782,86	
Encargado pruebas	Encargado prueba	20,00	0,15	1.000,00	152,38	
		Hombres mes	**3,87**	**Total**	**7.847,62**	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
HTC Wildfire	200,00	100	4	24	33,33
HTC Sensation	400,00	100	4	24	66,67
Nexus 4	349,00	50	3	24	21,81
					0,00
					0,00
					0,00
				Total	**121,81**

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
	Total	0,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas,

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	7.848
Amortización	122
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	1.594
Total	9.563

Ilustración 83: Presupuesto